UNIVERSITE SULTAN MOULAY SLIMANE, FACULTE DES SCIENCES ET TECHNIQUES

Centre d'Etudes Doctorales « Sciences et Techniques »
Formation doctorale «Mathématiques et Physiques Appliquesé »

MÉMOIRE

Présenté en vue d'obtenir

Le grade de

Docteur de l'Université Sultan Moulay Slimane

SPECIALITÉ : Mathématique-Informatique

Algorithmes Hiérarchiques de Résolution des Problèmes Décisionnels de Markov et des Modèles de Markov Cachés

Sanaa CHAFIK

Soutenue le 23 Février 2017 devant la commission d'examen :

Président	: Pr. Said MELLIANI.	PES	FST, Beni Mellal
Rapporteurs	: Pr, Abdelkrim HAQIQ.	PES	FST, Settat
	Pr, Abdelkrim MERBOUHA.	PES	FST, Beni Mellal
	Pr, Najlae IDRISSI.	PH	FST, Beni Mellal
Examinateur	: Pr, Mohamed FAKIR.	PES	FST, Beni Mellal

Directeur de thèse : Pr. Cherki DAOUI

Résumé

Cette thèse porte sur la résolution des problèmes de grandes tailles qui sont modélisés par des Modèles Markoviens tels que les Problèmes Décisionnels de Markov et les Modèles de Markov Cachés. Devant la limitation des algorithmes classiques de résolution de ces problèmes notre objectif est de chercher de nouvelles méthodes atténuant les problèmes générés dans le cas des grandes dimensions en réalisant un bon compromis entre qualité de la solution et accélération de la convergence avec un minimum de complexité.

Dans la première partie de ce travail, en utilisant la méthode de décomposition conçue aux Problèmes Décisionnels de Markov, nous proposons dans un premier temps, des algorithmes hiérarchiques pour résoudre quelques problèmes fondamentaux des Modèles de Markov Cachés en basant sur une modification au niveau de la méthode de décomposition. Dans un second temps, nous présentons une nouvelle version de l'algorithme d'Itération de la Valeur basée sur la méthode de décomposition et la méthode itérative d'accélération de convergence Gauss-Seidel. À la fin de cette partie, et pour montrer l'efficacité de nos approches, nous avons réalisé une étude détaillée de la complexité et de temps d'exécution résultants.

Vu l'importance du parallélisme dans le traitement des données de grandes tailles, dans la dernière partie nous avons débuté par l'introduction d'un procédé de parallélisation pour les problèmes subis de la décomposition, ce genre d'hybridation nous a permis de construire des algorithmes Hiérarchiques Parallèles. Par la suite, nous avons combiné une autre technique de décomposition qu'on a développée, dite décomposition topologique, avec le paradigme du parallélisme, ce qui nous a permis de concevoir des algorithmes Topologiques Parallèles.

Enfin, les expérimentations et les évaluations effectuées montrent que les algorithmes proposés, sont compétitifs avec les algorithmes de résolution classiques en termes de qualité, de complexité et de temps de convergence.

Mots-clés

Modèle Markovien, Problème Décisionnel de Markov, Modèle de Markov Caché, technique de décomposition, parallélisme, OpenMP, MPI, malédiction de la dimension, méthode itérative, Gauss-Seidel.

Abstract

This thesis focuses on solving problems of large sizes, which are modeled by Markov Model such as Markov Decision Problems and Hidden Markov Models. In front of the limitation of conventional algorithms dedicated to solving these problems, our objective is to search new methods attenuating the problems generated in the case of large dimensions and realize a good compromise between quality of the solution, accelerating convergence and minimal complexity.

In the first part, we benefited of the prerogatives of the decomposition method designed to Decisional Markov Model. We proposed initially, the hierarchical algorithms to solve the fundamental problems of the Hidden Markov Models basing on a modification in the decomposition method. Secondly, we described a new class of Value Iteration algorithm and which benefits of the hierarchization provided by the decomposition method and the convergence acceleration methods such as Gauss-Seidel. At the end of this part, and to show the effectiveness of our approach, we conducted a detailed study of the complexity and the resulting execution time.

In the last part, we adopted the parallelism as a basic development tool given its importance in the treatment of large size of data. We started with the introduction of a parallelization process derived from the adopted decomposition technique, this type of hybridization have allows us to construct the Hierarchical Parallel accelerated algorithms. Thereafter, we proceeded to another sort of hybridization that combines a decomposition technique developed called topological decomposition with the paradigm of parallelism; it allows us to create a Parallel Topological algorithms.

Finally, the experiments carried out show that the proposed algorithms are competitive with conventional resolution algorithms in terms of quality, complexity and convergence time.

Keywords
Markov Model, Markov Decision Problem, Hidden Markov Model, Decomposition technique, parallelism, OpenMP, MPI, curse of dimensionality, iterative method, Gauss-Sidel.

Remerciements

D’abord, je tiens à remercier spécialement mon encadrant : Monsieur Cherki DAOUI de la Faculté des Sciences et Techniques de l’Université Sultan Moulay Slimane qui m’a aidé tout au long de cette thèse. Tout en me laissant une grande liberté dans les directions que je souhaitais prendre. Cette thèse n’aurait jamais vu le jour sans son aide en termes d’idées, de conseils et de discussions fructueuses, je le remercie vivement. Ce fut un réel plaisir de l’avoir comme encadrant de thèse.

Je tiens à remercier Monsieur Abdelkrim HAQIQ Monsieur Abdelkrim MERBOUHA et Madame Najla IDRISSI d’avoir rapportés sur cette thèse, d’avoir contribués à améliorer la qualité de ce mémoire et du temps qu’ils ont consacré à la lecture du manuscrit.

Je tiens à remercier Monsieur Said MELLIANI et Monsieur Mohamed FAKIR d’avoir acceptés de participer au jury de cette thèse.

Je remercie du fond du cœur ma mère et mon père qui ont été présents pendant tout le cursus de mes études. Leur apport est inestimable. Que Dieu les récompense.

Je tiens aussi à remercier mon frère Oussama, ma sœur Siham et mes proches qui m’ont toujours soutenu et aidé dans mes choix, ils n’ont jamais cessé de m’encourager durant mes études.

Finalement, je tiens à remercier tous mes amis et collègues qui ont contribué de près ou de loin à l’accomplissement de ce travail.

Sommaire

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Les Modèles Markoviens

Notre domaine de recherche est les Modèles Markoviens qui représentent une collection de processus stochastiques permettant de modéliser et d'étudier des systèmes dynamiques régis par des lois de probabilité vérifiant la propriété de Markov, qui suppose que l'évolution du système dans le futur ne dépend que de l'état actuel.

Selon l'observabilité des états du système, nous distinguons quatre Modèles Markoviens : les Chaines de Markov, les Problèmes Décisionnels de Markov, les Modèles de Markov Cachés et les Problèmes de Décision Markoviens Partiellement Observables.

Problèmes Décisionnels de Markov

Les Problèmes Décisionnels de Markov représentent le cadre naturel de modélisation des problèmes de décision séquentielle dans l'incertain. Ils sont construits à partir des Chaines de Markov dont la dynamique du système est contrôlée par des décisions et évaluée par des revenus. Ce type de modèles est utilisé dans de nombreux domaines d'application, de l'économie au domaine des systèmes de file d'attente en passant par la robotique ou domaine médical [18, 22, 26, 63, 64, 74]. Les solutions sont générées sous forme de stratégies ou de politiques qui spécifient l'action à entreprendre en chaque instant pour les positions possibles du décideur. Pour la résolution des Problèmes Décisionnels de Markov, Il existe deux grandes familles de méthodes issues de la programmation dynamique : les méthodes itératives telles que l'algorithme d'Itération de la Valeur et l'algorithme d'Itération de la Politique et les méthodes directes fondées sur la programmation linéaire.

L'algorithme d'Itération de la Valeur [36] est l'un des algorithmes standards pour la résolution des Problèmes Décisionnels de Markov, il est basé sur la résolution directe de l'équation d'optimalité de Bellman, il améliore itérativement la fonction valeur jusqu'à la convergence à la solution recherchée. À l'inverse, l'algorithme d'Itération de la Politique améliore les politiques elles-mêmes.

Modèles de Markov Cachés

Les Modèles de Markov Cachés représentent une extension des Chaines de Markov, dont on ajoute la non observabilité des états du système qui peut être identifiée seulement à partir des données observées. Ce modèle est massivement utilisé surtout dans le domaine de la reconnaissance de textes manuscrits [11, 69, 84], reconnaissance de la parole et d'images [71, 57, 34] et l'analyse de séquences biologiques [60].

Les trois problèmes fondamentaux : l'évaluation, le décodage et l'apprentissage sont résolus en utilisant les algorithmes classiques du Modèle de Markov Caché tels que l'algorithme Forward-Backward, l'algorithme Viterbi et l'algorithme Baum-Welch.

Le travail que nous présentons dans cette thèse se focalise sur ces deux Modèles Markoviens : les Problèmes Décisionnels de Markov et les Modèles de Markov Cachés.

Notre problématique

L'utilisation des Modèles Markoviens comme cadre de modélisation et de résolution de la plus part des problèmes réels aboutit souvent à des espaces d'états multidimensionnels. Ainsi, les algorithmes de résolution classiques qui apparaissent efficaces en faible dimension vont se révéler peu performants en grande dimension. Cette problématique est connue d'après Bellman par le terme de la « malédiction de la dimension », elle relate un facteur principal limitant considérablement la résolution numérique et donc la portée pratique des Modèles Markoviens.

Parmi les solutions proposées pour remédier à cette problématique nous pouvons citer la méthode de décomposition séquentielle proposée par Abbad, Boustique et Daoui [1, 2, 27, 28]. Elle est inspirée de la technique algorithmique « Diviser pour régner » et du concept de la théorie des graphes (Hiérarchisation des sommets et hiérarchisation des niveaux). Cette méthode vise à décomposer le problème initial en des petits problèmes, les résoudre par les algorithmes classiques et ensuite reconstituer une solution globale. Elle commence par partager l'espace d'états en composantes fortement connexes, ces classes sont ensuite réparties dans des niveaux à partir desquelles des petits problèmes dits restreints sont construites et résolus séparément par l'un des algorithmes classiques.

La technique du parallélisme est un procédé suffisamment connu permettant d'atténuer le fardeau de calcul pour des problèmes de grandes dimensions. Elle repose sur le même principe qui consiste à décomposer le problème global en parties distinctes qui peuvent être

résolus simultanément sur plusieurs processeurs, ce qui provoque une exécution accélérée de l'algorithme par apport à la version séquentielle.

Dans la suite nous allons décrire brièvement le contenu de cette thèse.

Plan synthétique de la thèse

Le présent document se décompose en deux grandes parties. La première partie réalise un état de l'art des différents concepts de base utilisés pendant l'évolution de nos travaux ainsi que le problème que nous allons traiter. La seconde partie présente nos contributions pour la résolution de la problématique soulevée sous forme d'algorithmes hiérarchiques proposés. Enfin, en vue de valider nos approches, des études de la complexité et différentes expérimentations sont menés.

Partie I : Etat de l'art

Nous appuyant sur une étude bibliographique, dans le premier chapitre, nous identifierons notre domaine de recherche à savoir les Modèles Markoviens et leurs différentes classifications : les Chaines de Markov, les Problèmes Décisionnels de Markov, les Modèles de Markov Cachés et les Problèmes de Décision Markoviens Partiellement Observables. Nous préciserons la problématique commune entre ces modèles qui réside dans le cas de grandes dimensions. Le deuxième chapitre, portera sur une description détaillée des Modèles de Markov Cachés et leurs problèmes fondamentaux, et après avoir présenté les algorithmes de résolution nous passerons au chapitre suivant, qui s'attachera au deuxième modèle, les Problèmes Décisionnels de Markov, nous exposerons leur cadre formel ainsi que leur classe de complexité. Nous présenterons ensuite les algorithmes de résolution et plus précisément l'algorithme d'Itération de la Valeur et l'algorithme d'Itération de la Politique. Nous finaliserons ce chapitre en évoquant les différents travaux de littérature qui permettent d'alléger notre problématique. Nous consacrerons le dernier chapitre au paradigme de la programmation parallèle qui représente plus qu'un outil d'accélération, mais une solution palpable pour les problèmes de grandes tailles. Ce chapitre éclaircira les différents types du parallélisme existants et quelques métriques pour l'évaluation des performances de la solution parallèle.

Partie II : Contributions

La seconde partie de ce document s'articulera sur nos contributions, elle est subdivisée en quatre chapitres. Les trois premiers chapitres utiliseront la méthode de décomposition conçu

pour les Problèmes Décisionnels de Markov [1, 2, 27, 28]. Le premier chapitre présentera cette méthode en détails. Dans le deuxième chapitre, nous allons modifier cette méthode de décomposition pour l'appliquer aux Modèles de Markov Cachés en proposant trois algorithmes hiérarchiques : Forward Hiérarchique, Backward Hiérarchique et Baum-Welsh Hiérarchique, nous allons à la fin exposer les résultats qui valident l'efficacité de ces algorithmes.

Le troisième chapitre portera sur une combinaison entre la méthode de décomposition et la méthode itérative Gauss-Seidel, ce qui représente un moyen d'accélération de la convergence de l'algorithme d'Itération de la Valeur.

Le dernier chapitre, traitera l'hybridation de la technique du parallélisme sur la méthode de décomposition utilisée ainsi que sur une autre décomposition topologique que nous avons proposée. Il détaillera l'idée conceptuelle du parallélisme et les différents classements de ces architectures existantes, et éclaircira plus l'importance de son application aux Problèmes Décisionnels de Markov et sur des algorithmes. Enfin, il passera à l'expérimentation et l'évaluation des algorithmes Hiérarchiques, Topologiques et Parallèles proposés avec leur validation, l'étude de leur complexité et avec la comparaison du temps de calcul résultants.

Liste des publications parues ou à paraître

- Chafik, S., & Cherki, D. (2013). Some Algorithms for Large Hidden Markov Models. World Journal Control Science and Engineering, 1(1), 9-14.
- Chafik, S., & Cherki, D. Hierarchical Algorithm for Hidden Markov Model. International Journal of Advanced Computer Science and Applications. www. thesai. org| info@ thesai. org.
- Chafik, S., & Daoui, C. (2015). A Modified Value Iteration Algorithm for Discounted Markov Decision Processes. Journal of Electronic Commerce in Organizations (JECO), 13(3), 47-57.
- Chafik, S., & Daoui, C. (2016). A Modified Policy Iteration Algorithm for Discounted Reward Markov Decision Processes. International Journal of Computer Applications, 133(10), 28-33.
- Chafik, S., Daoui, C. & Nachaoui, M. A Parallelized Hierarchical Value Iteration Algorithm for Discounted Markov Decision Processes. (Soumis à: Applied and Computational Mathematics) ;
- Chafik, S., Larach, A. & Daoui, C. Parallel Hierarchical Pre-Gauss-Seidel Value Iteration Algorithm. (Soumis à: International Journal of Decision Support System Technology) ;
- Larach, A., Chafik, S., & Daoui, C. Accelerated Decomposition Techniques for Large Discounted Markov Decision Processes. (Soumis à: journal of industrial engineering international).

Liste des communications dans des conférences internationales et nationales

- Hiearchical Algorithm for Hidden Markov Model. La 3ème édition du Symposium International de Traitement Automatique de la Langue et Culture Amazigh, FST, Beni Mellal, Maroc, 2-4 Mai 2013;
- L'amélioration de l'algorithme Forward. Premier Workshop en Imagerie, Systèmes et Applications, Faculté Polydisciplinaire, Taza, Maroc, 23-24 Mai 2013;
- Some Algorithms for Large Hidden Markov Models. First International Conference on Business Intelligence. CBI'14, 29-30 Avril 2014, Beni Mellal;
- Modified Value Iteration Algorithm for Markov Decision Processes. 6 ème édition des journées Doctorales en Technologies de l'information et de la Communication JDTIC'14, 19-20 Juin 2014, ESIAS, Rabat;
- Parallelizing a Modified Value Iteration Algorithm for Solving Markov Decision Processes. Aux Journées Doctorales JDoc'15 Du 26-28 Mars 2015, Beni Mellal;
- A Modified Value Iteration Algorithm for Discounted Markov Decision Processes. International Conference on Business Intelligence. CBI'15, 23-25 Avril 2015, Beni Mellal;
- A Parallelized Hierarchical Value Iteration Algorithm for discounted Markov Decision Processes. Numerical Analysis and Optimization Days JANO11 du 27-29 Avril 2016, Beni Mellal.

Partie I : Etat de l’art

Introduction de la partie I

L'objectif de cette première partie est d'introduire les concepts qui seront à la base de ce travail. Nous commencerons, dans le premier chapitre par définir les **Modèles Markoviens** puis, nous décrirons leurs extensions et aussi leurs faiblesses qui résident dans le cas de **grandes dimensions**. Le deuxième chapitre portera sur la formalisation des **Modèles de Markov Cachés** et leurs algorithmes conçus pour résoudre quelques problèmes fondamentaux. Le troisième chapitre traitera les **Problèmes Décisionnels de Markov** et les différentes techniques réalisées pour développer ce modèle. Le **parallélisme** représente une technique de calcul sollicitée pour les problèmes de grandes tailles, le quatrième et dernier chapitre de cette partie traitera, ce concept et expliquera l'importance de son application dans le domaine des problèmes de grandes tailles, notamment pour les Problèmes Décisionnels de Markov.

1 Modèles Markoviens (MMs)

Dans ce chapitre, nous allons nous focaliser sur les **Modèles Markoviens** (MMs) qui représentent une collection de modèles stochastiques permettant de modéliser et d'étudier des systèmes régis par des lois de probabilité et encadrés par la **propriété de Markov**. En commençant par les **Chaines de Markov** (CMs) qui décrivent la forme basique des autres catégories de MM tels que les **Problèmes Décisionnels de Markov** (PDMs), les **Modèles de Markov Cachés** (MMCs) et les **Problèmes de Décisions Markoviens Partiellement Observables** (PDMPOs).

Avant d'entrer dans la phase de résolution au chapitre suivant, nous allons présenter la théorie de la **complexité algorithmique** qui représente le calibre par lequel nous pouvons évaluer de manière théorique la quantité d'appels aux opérations, ainsi que l'efficacité des algorithmes de résolution.

Dans la dernière partie de ce chapitre, nous allons discuter la difficulté de la résolution de ces problèmes modélisés par ces modèles dans le cas de grandes tailles qui relate un facteur principal limitant considérablement l'utilisation des MMs.

1.1 Extensions du Modèle Markovien

Le mathématicien russe Andrei Andreyevich Markov (1856-1922) a étudié des systèmes dont l'évolution dans le temps est incertaine au tout début du XXe siècle, ces systèmes sont des automates stochastiques qui seront à l'origine de la théorie des processus stochastiques. Ces derniers sont des familles de variables aléatoires S_t, représentant l'état du processus, elles sont indexées par un paramètre t qui prend ses valeurs dans l'ensemble T représentant le temps. La variable S_t représente l'état du processus au temps t et l'ensemble de toutes les valeurs possibles pour cette variable est appelé l'espace des états du processus et sera noté S. Le processus est en temps continu si T est continu (e.g., T = [0 , ∞[), et en temps discret si T est discret (e.g., T ={0, 1, 2, . . .}). Lorsque S est fini ou dénombrable, le processus est appelé une chaine. Andrei Andreyevich Markov s'est particulièrement intéressé aux systèmes possédant la propriété dite de Markov (1.1) qui suppose que la connaissance de l'état d'un système à l'instant t est suffisante pour prédire l'évolution du système à l'instant suivante t+1, il ne nécessite pas la connaissance du passé. Cette propriété peut être résumée par l'équation suivante:

$$P(S_{t+1}=s_{t+1}\,|\,S_t=s_t, S_{t-1}=s_{t-1},...,S_0=s_0)=P(S_{t+1}=s_{t+1}\,|\,S_t=s_t) \qquad (1.1)$$

Cette propriété peut paraître très précise, ce qui nécessite une bonne construction des états du système de telle sorte qu'ils contiennent toutes les informations nécessaires à la prédiction de l'évolution du système.

Nous allons s'intéresser aux MMs à espace temps et état discrets. Ils existent plusieurs extensions des MMs: les CMs, les MMCs, les Modèles de Décisions Markoviens (ou bien les PDMs) et les Modèles de Décisions Markoviens Partiellement Observables (ou bien les PDMPOs), montrés par le diagramme de la figure 1.1.

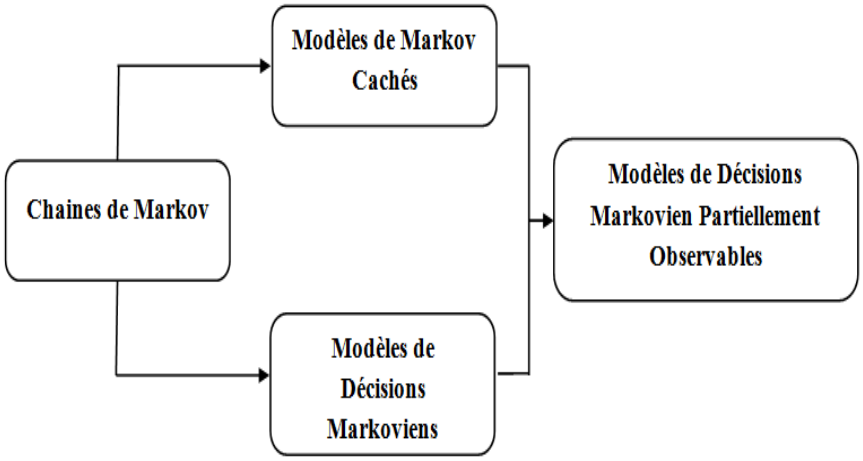

Figure 1.1- Modèles Markoviens discrets

1.1.1 Chaine de Markov

Une CM c'est le MM de base correspond à un simple graphe d'états. Elle permet de représenter la dynamique d'un système dont l'évolution est régie par des lois stochastiques connues d'avance, satisfaisante la propriété de Markov (1.1), dotée d'un ensemble des états S totalement observable et d'une fonction de transition probabiliste P, tel que :

$$P : S \times S \rightarrow [0,1] \equiv p(s'|s), \forall s, s' \in S \qquad (1.2)$$

A chaque instant, le modèle subit une transition qui va totalement changer son état actuel s vers un autre état s' avec une probabilité de passage $p(s'|s)$. Les CMs trouvent des applications dans beaucoup de domaines comme la génétique, la physique, la recherche opérationnelle, etc.

Considérons une souris qui se déplace dans un labyrinthe représenté sur la figure 1.2. Il comporte 6 compartiments ou bien des états numérotés de 1 à 6. On suppose qu'elle change de compartiment à chaque instant t = {0, 1, 2, 3…}. A chaque changement d'état, le nouvel état est choisi avec une distribution de probabilité équiprobable et ne dépend que de l'état actuel de la souris.

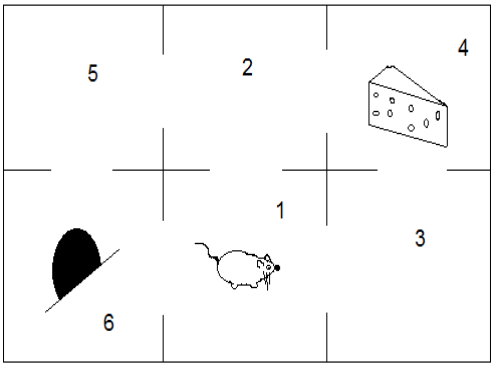

Figure 1.2- Exemple de labyrinthe d'une souris

La souris se déplace selon la matrice suivante :

$$P=\begin{pmatrix} 0 & 1/3 & 1/3 & 0 & 0 & 1/3 \\ 1/3 & 0 & 0 & 1/3 & 1/3 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 0 & 1/2 & 0 \end{pmatrix}$$

Alors la souris passe dans sa tanière par une seule transition avec une probabilité de 1/3, sinon la souris va choisir de passer soit dans le compartiment 3 avec une probabilité 1/3, soit dans le compartiment 2 avec une probabilité 1/3. Depuis le compartiment 3 elle a une chance sur deux de trouver la nourriture et depuis le compartiment 2 elle a une chance sur trois de trouver la nourriture. Il y a donc une probabilité de 1/6 (Voir figure 1.3) en passant par la case 3, et 1/9 (Voir figure 1.4) en passant par la case 2 pour que la souris trouve la nourriture au bout de deux minutes. Dans les autres cas, elle se retrouve dans le compartiment initial, ce qui permet d'établir une formule de récurrence pour les probabilités cherchées.

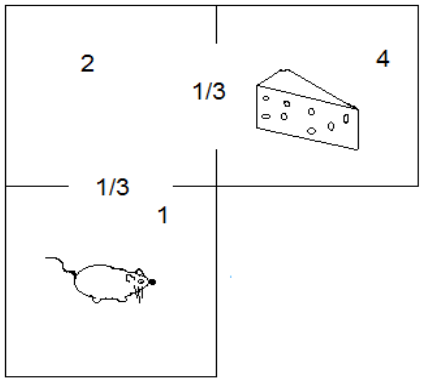

Figure 1.3- Premier choix de déplacement de la souris

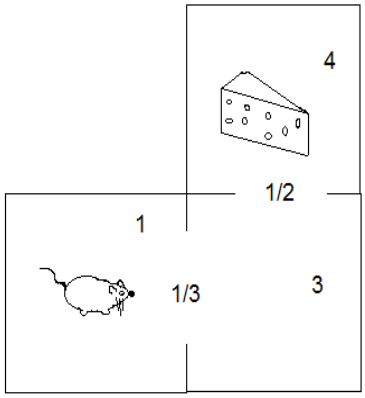

Figure 1.4- Deuxième choix de déplacement de la souris

1.1.2 Modèle de Markov Caché

Contrairement aux CMs, les MMCs modélisent des environnements des systèmes partiellement observables où l'agent n'a accès qu'à une observation partielle de son état ou bien une connaissance partielle. Les MMCs se basent sur deux processus stochastiques, un processus pour les états cachés et l'autre pour les observations. Ainsi, en plus de l'ensemble des états S et la fonction de transition P des CMs, le MMC est composé d'un ensemble d'observations (ou bien symboles) O et d'une fonction d'observation B, comme il est illustré au figure 1.5.

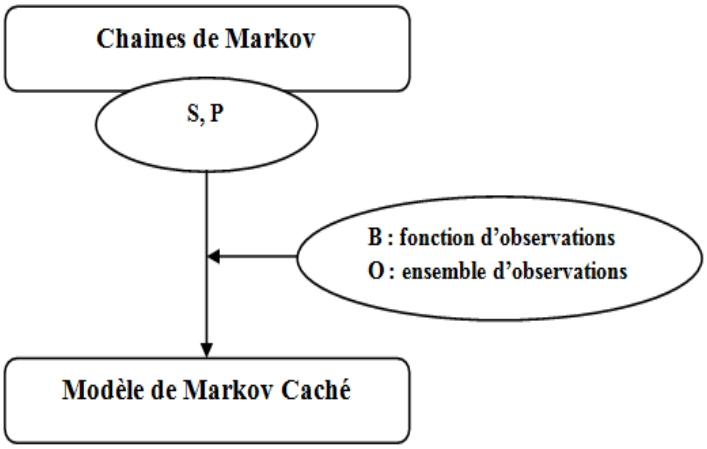

Figure 1.5- Relation entre les CMs et les MMCs

1.1.3 Problème de Décision Markovien

Comme le montre la figure 1.6, les PDMs sont des CMs dont l'agent peut influencer l'évolution du système totalement observable par l'introduction de la notion d'action et de récompense dans son cadre formelle. Les PDMs contiennent donc, en plus de l'ensemble d'états S un ensemble fini d'actions A et une fonction de récompense R. L'ensemble d'actions A permet à l'agent d'intervenir sur le système en changeant la définition de la fonction de

transition $P : S \times A \times S \rightarrow [0;1] \equiv p(s'|s,a)$, elle indique la probabilité de passage vers un état s', en partant d'un état donné s avec $s, s' \in S$; en connaissant l'action en cours d'exécution a, a∈A, cette fonction est représentée par |A| matrices, chacune décrivant l'évolution du système pour une action précise. La fonction de récompense R, permet de guider le comportement de l'agent pour lui dire ce qu'il faut faire et ce qu'il ne faut pas, en affectant à chaque comportement une valeur numérique réel traduisant la qualité d'un comportement donné (1.3), (1.4) et (1.5). Cette fonction peut être définie de plusieurs façons :

$$R : S \times A \rightarrow \mathbb{R} \tag{1.3}$$

$$R : S \rightarrow \mathbb{R} \tag{1.4}$$

$$R : S \times S \rightarrow \mathbb{R} \tag{1.5}$$

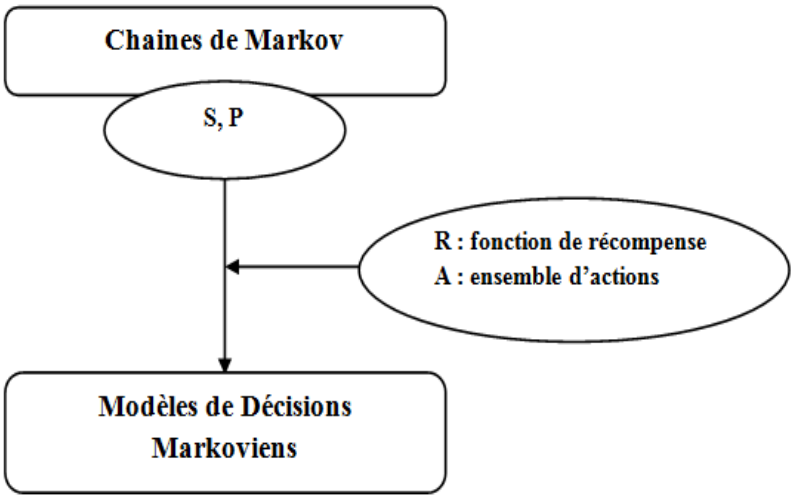

Figure 1.6- Relation entre les CMs et les PDS

Les MMCs et les PDMs sont deux sortes d'évolution des CMs sur lesquels nous allons nous focaliser tout le long de ce mémoire. Comme pour les CMs, chaque état des deux modèles, PDM et MMC, doit contenir suffisamment d'informations pour permettre une prédiction parfaite du comportement du système.

1.1.4 Modèle de Décision Markovien Partiellement Observable

Les Modèles de Décisions Markoviens Partiellement Observables (ou bien les Problèmes de Décisions Markoviens Partiellement Observables PDMPOs) [25] sont issus de la combinaison des PDMs et des MMCs. Ce modèle inclue donc simultanément la notion d'observabilité partielle et la notion d'influence sur l'évolution du système, la figure 1.7 résume cette fusion.

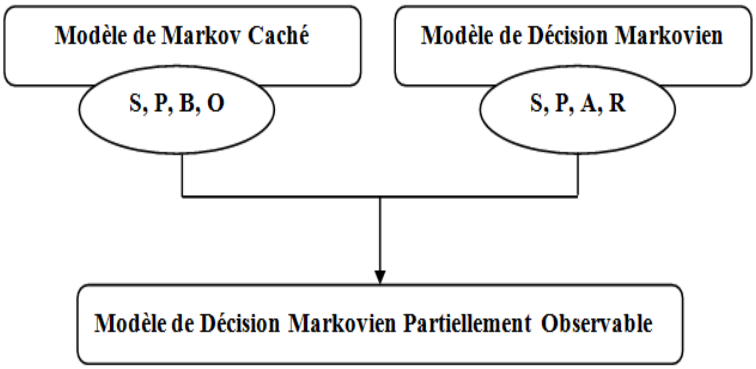

Figure 1.7- Relation entre les POPDMs et les autres Modèles Markoviens

1.2 Etude de complexité

Au début de la découverte des algorithmes fondamentaux tels que le tri rapide, l'algorithme de Kruska, etc. On se contentait de dire qu'un algorithme se déroule en tel unité de temps avec des données de taille aléatoires et sur un tel ordinateur, cette démarche rendait difficile la comparaison des algorithmes entre eux. Au cours des années soixante et devant la lenteur et la faible capacité de mémoire des ordinateurs de l'époque (Figure 1.8), les chercheurs se sont intéressés donc naturellement à l'efficacité des algorithmes, ils ont par exemple cherché les algorithmes qui tournent sur ce genre de machine sans que le résultat mette beaucoup de temps à arriver.

Figure 1.8- Machine utilisé en 1948 : IBM SSEC

s.d. Image libre de droits

Afin de comparer l'efficacité de plusieurs algorithmes qui peuvent répondre à un objectif fixé, on utilise une mesure que l'on appelle la complexité. L'analyse de la complexité d'un problème est une branche de l'informatique théorique qui est concentrée sur toutes genres d'études touchant les fondements logiques et mathématiques de l'informatique.

L'analyse de la complexité d'un problème permet d'étudier l'espace mémoire et le temps requis par toutes sortes d'algorithmes résolvant ce problème. Ce qui revient à classifier le comportement de ces algorithmes lorsqu'ils sont utilisés surtout sur des données de grandes tailles. Cette analyse est étroitement associée à un modèle de calcul, le model proposé par Alan Turing en 1936 [99] reste le modèle le plus utilisé. Généralement, le calcul dans tous les modèles qui existent dans la littérature est formé par un ensemble d'étapes élémentaires, devant chacune de ces étapes, on choisit une action à exécuter parmi un ensemble d'actions pour un état donné de la mémoire de la machine.

Dans ce sens on différencie entre deux types de machines, les machines déterministes qui représentent le modèle classique d'une machine (le cas de nos ordinateurs) dont l'action élémentaire à effectuer pour un état donné de la machine est toujours entièrement déterminée. S'il peut y avoir plusieurs choix possibles d'actions à effectuer, on est devant l'autre type des machines dites non déterministes, ces dernières sont plus efficaces et peuvent trouver une solution aux problèmes posés en un temps raisonnable alors que les machines déterministes ne savent pas les résoudre à moins de prendre des années.

La complexité d'un algorithme se mesure en calculant le nombre d'opérations élémentaires telles que le nombre de comparaisons et le nombre d'opérations arithmétiques pour traiter un problème de taille n. On désigne par « les classes de complexité » des ensembles de problèmes qui ont la propriété d'avoir tous la même complexité, c'est une sorte d'hiérarchie de difficultés entre les problèmes algorithmiques. Les classes les plus connues sont des classes définies sur des machines de Turing avec des contraintes de temps et d'espace.

1.2.1 Complexité en temps

On distingue quatre familles principales de classes de complexité suivant le temps :

- La classe P-complet : des problèmes qui peuvent être résolus en temps polynomial par une machine de Turing déterministe. C'est la classe des problèmes facilement solubles;
- La classe NP-complet : ce sont des problèmes résolubles en temps Polynomial par une machine Non déterministe (Et non pas Non polynomial), c'est la classe des langages reconnus par une machine de Turing non déterministe polynomiale;
- EXPTIME : la classe des problèmes qui peuvent être résolus en temps exponentiel sur une machine déterministe;
- NEXPTIME : la classe des problèmes qui peuvent être résolus en temps exponentiel sur une machine non-déterministe.

1.2.2 Complexité en espace mémoire

La complexité d'un problème peut également être évaluée en termes d'espace ce qui représente une ressource critique dans les algorithmes sur tout quand on parle d'un algorithme qui requiert de la mémoire supplémentaire :

- La classe L : la classe des problèmes qui peuvent être résolus en espace logarithmique sur une machine déterministe;
- La classe NL : la classe des problèmes qui peuvent être résolus en espace logarithmique sur une machine non-déterministe;
- PSPACE : la classe des problèmes qui peuvent être résolus en espace polynomial sur une machine déterministe;
- EXPSPACE : la classe des problèmes qui peuvent être résolus en espace exponentiel.

1.3 Problèmes de grandes tailles

Depuis 1980, la production de données numériques double tous les trois ans ce qui atteint aujourd'hui le chiffre de 2×10^{21} octets par jour [95]. En raison de cette immense augmentation des données dans plusieurs domaines, une initiative de recherche au sens du Big Data a été provoquée.

Les algorithmes stochastiques visant à résoudre des problèmes d'optimisation ou d'estimation complexes tels que les algorithmes utilisés dans la résolution des problèmes modélisés par les MMs sont aussi caractérisés par l'exploration des espaces de grandes dimensions. Ainsi, la grande dimension représente l'un des difficultés majeures qui explique l'incapacité de ces algorithmes. En pratique, le traitement des données devient très difficile dès que la taille du modèle s'approche ou arrive à 10^6. Le terme malédiction de la dimension ou le fléau de la dimension (Curse of dimensionality) est révélé par le mathématicien américain Richard Bellman en 1961 [10]. Il explique que les problèmes de grandes dimensions sont très difficiles à résoudre, même en utilisant des algorithmes d'optimisation qui visent à résoudre les problèmes difficiles issus des domaines de la recherche opérationnelle, de l'ingénierie, de l'intelligence artificielle, etc.

La problématique de grande dimension a affecté plusieurs domaines tels que l'apprentissage et la recherche par similarité [12]. Ainsi, si on traite des données de grandes tailles, les algorithmes qui apparaissent efficaces en faible dimension vont se révéler peu performants en grande dimension. Beyer [14] en 1999, a exploré l'effet de la dimensionnalité sur la méthode d'apprentissage supervisé : la méthode de « Plus proche voisin ». Il a démontré que plus le nombre de dimensions est grande, plus le rapport entre la distance minimale et la distance maximale entre les données tend vers un, ce qui est contradictoire avec la notion de Plus proches voisins.

Poļaka [76,77] a étudié et remédié à la problématique de la malédiction de la dimension dans le domaine de la bioinformatique et l'analyser de la matière biologique au niveau de la classification des données.

1.4 Conclusion

Dans ce chapitre, nous avons présenté les différentes extensions des **Modèles Markoviens** (MMs). Nous avons exposé un bref rappel sur la **complexité** et ses différentes classes dont on aura besoin dans la suite pour aborder et comparer **les algorithmes classiques ou améliorés** utilisés pour la résolution des problèmes modélisés par ces modèles. Nous avons ensuite mis en avant l'aspect limitatif des MMs du point de vue pratique dans le cas de **grandes dimensions**.

2 Modèles de Markov Cachés (MMCs)

Après avoir précisé notre domaine de recherche et ses principales limites de point de vue pratique, nous allons traiter en détail dans ce chapitre les **Modèles de Markov Cachés** (MMCs) qui représentent une autre évolution possible des Chaines de Markov (CMs). Ils sont utilisables dans plusieurs domaines tels que : la biologie moléculaire, le traitement de signal et la reconnaissance de la parole. Nous allons ensuite décrire les algorithmes classiques pour résoudre les problèmes fondamentaux des MMCs : l'algorithme **Viterbi**, l'algorithme **Forward-Backward** et l'algorithme **Baum-Welsh**.

2.1 Cadre formelle des MMCs

Les MMCs sont une extension des MMs introduites par Baum et ses collaborateurs dans les années 1960-70s [8], ils représentent un modèle statistique qui permet la représentation d'un processus de Markov dont l'état est caché ou non observable. Cependant, l'information sur ces états cachés peut être déduite de la donnée observée.

Le domaine de traitement de la parole dans les années 70 ; était le premier domaine auquel les MMCs ont été appliqués, ils sont devenues leur modèle de référence, plusieurs techniques ont été développées pour le cadre applicative des MMCs. Ces techniques furent ensuite adaptées aux problèmes de la reconnaissance de textes manuscrits, l'analyse de séquences biologiques, la reconnaissance d'images, etc. Les MMCs sont basés sur des algorithmes d'estimation statistiques tels que l'algorithme Forward, Backward et Viterbi, l'utilisation de ces trois algorithmes permet de résoudre les trois problèmes fondamentaux détaillés dans la section suivante. Le MMC noté λ peut être défini par le quintuple λ=(S, O, A, B, Π) :

- Un ensemble d'état $S = \{s_1, s_2, ..., s_{|S|}\}$, à chaque instant t l'état du système est noté par S_t qui est une variable aléatoire à valeur dans l'ensemble d'observation S;
- Un ensemble d'observations distinctes selon T symboles $O = \{o_1, o_2, ..., o_T\}$, à chaque instant t l'observation est notée par O_t qui est une variable aléatoire à valeur dans l'ensemble d'observation O;
- La matrice des probabilités de transition A=$\left[a_{ij}\right]$, avec $a_{ij} = p(S_{t+1} = s_j \mid S_t = s_i)$ représente la probabilité de passer de l'état s_i à l'état s_j ;
- La matrice des probabilités d'observation (Ou d'émission) $B = \left[b_j(m)\right]$, avec $b_j(m) = p(O_t = o_m \mid S_t = s_j)$ représente la probabilité d'émettre le symbole o_m à l'instant t par l'état s_j ;

- La matrice des probabilités initiales $\Pi=[\pi_i]$, avec $\pi_i = p(S_0 = s_i)$ représente la probabilité que le processus démarre à l'instant t=0 à partir de l'état s_i, nous avons $\sum_{i=1}^{N} \pi_i = 1$.

Pour un système météorologique (Figure 2.1) à deux états cachés s_1 = Pluie et s_2 = Soleil et un ensemble d'observations ou d'activités O={o_1=Marche, o_2=Etude, o_3=Sport, o_4 =Sommeil} que la personne peut faire en fonction de l'atmosphère qu'il fera.

La probabilité d'émettre l'un des symboles de O (Ou bien faire l'un des activités) sachant que le jour est ensoleillé ou le jour est pluvieux :

b_{pluie} (Marche)=0.1, b_{pluie} (Sport)=0.2, b_{pluie} (Etude)=0.3, b_{pluie} (Sommeil)=0.4

b_{soleil} (Marche)=0.3, b_{soleil} (Sport)=0.4, b_{soleil} (Etude)=0.2, b_{soleil} (Sommeil)=0.1

La probabilité de transition $A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$, et une probabilité d'initialisation, on a : $\pi_{pluie} = 0.6$ et $\pi_{soleil} = 0.4$.

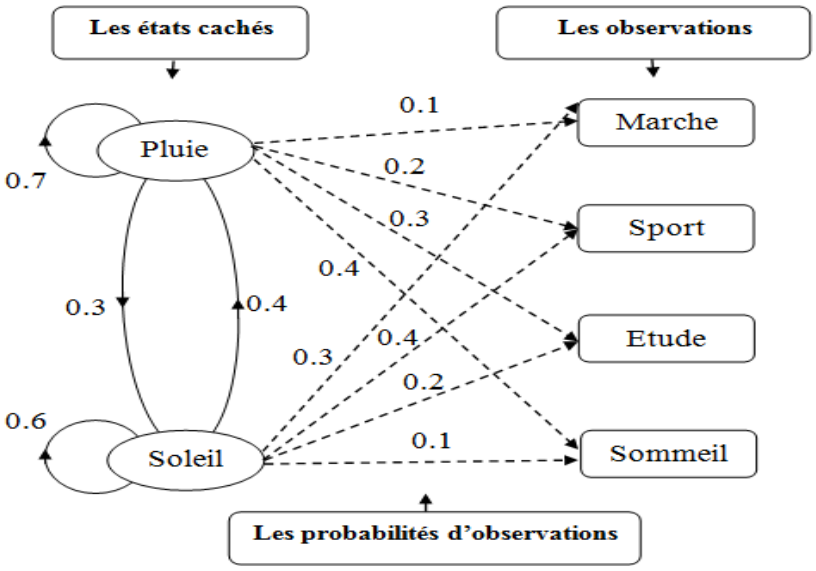

Figure 2.1- Exemple d'un MMC : le système météorologique

2.2 Problèmes fondamentales

Les MMCs trouvent leur intérêt dans la résolution de trois problèmes fondamentaux qui sont l'évaluation, le décodage et l'apprentissage [83, 100]. Soit $O = \{o_1, o_2, ..., o_T\}$; une séquence d'observations où T est le nombre d'observations.

2.2.1 Evaluation

Etant donnés un MMC : $\lambda = \{\Pi, A, B\}$, comment peut-on calculer efficacement la probabilité (La vraisemblance) que la suite d'observations O soit produite par λ : $p(O \mid \lambda)$. Autrement dit, comment évaluer le modèle afin de choisir parmi plusieurs celui qui génère le mieux la suite d'observations O. Plusieurs techniques permettent de résoudre ce problème ; on trouve la méthode d'évaluation directe et la procédure « Forward » ou « Backward ».

2.2.2 Décodage

Le problème de décodage ou de reconnaissance, il précise l'estimation de la suite d'états cachés $\{s_1, s_2, ..., s_{|S|}\}$ appartenant à S; sachant qu'on a l'ensemble d'observations O et le modèle $\lambda = \{\Pi, A, B\}$. En fait, Il s'agit de trouver dans le modèle la suite d'états qui maximise la probabilité $p(O, S \mid \lambda)$. L'algorithme le plus connu pour résoudre ce problème est l'algorithme Viterbi.

2.2.3 Apprentissage

L'apprentissage d'un MMC consiste à estimer le vecteur paramètre $\lambda = \{\Pi, A, B\}$ sur la base d'un ensemble de séquences d'observations appelé corpus d'apprentissage en se fondant cette fois-ci sur la notion de maximum de vraisemblance qui est une stratégie efficace pour inférer les paramètres. Il n'existe pas un algorithme qui donne des résultats exacts mais l'utilisation de vraisemblance donne des résultats satisfaisants en utilisant l'algorithme Baum-Welch.

2.3 Algorithmes classiques

Dans ce qui suit, nous considérons une séquence d'observations O à T observations: $O = \{o_1, o_2, ..., o_T\}$.

2.3.1 Algorithme Forward-Backward

Nous allons présenter l'algorithme Forward-Backward qui permet de résoudre le problème d'évaluation, pour calculer la probabilité d'engendrer la séquence d'observation O étant donné le modèle λ, nous faisons recours à deux procédures : Forward $\alpha_t(i)$ et Backward $\beta_t(i)$.

La variable Forward $\alpha_t(i)$ (2.1) est une variable associée à chaque paire (i, t), elle désigne la probabilité d'avoir généré la séquence d'observations partielle $O = \{o_1, o_2, ..., o_t\}$ en arrivant à l'état s_i à l'instant t, étant donné le modèle λ :

$$\alpha_t(i) = p(O_1 = o_1, ..., O_t = o_t, S_t = s_i \mid \lambda) \qquad (2.1)$$

Le calcul de la variable Forward représente la base de l'algorithme Forward comme il est précisé ci-dessous :

Algorithme 2.1- Forward

Etape 1 : Initialisation, pour tout $i \in [1,|S|]$:

$$\alpha_1(i) = \pi_i \times b_i(o_1)$$

Etape 2 : Induction, pour tout $t \in [1,T-1]$ et $j \in [1,|S|]$:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{|S|} \alpha_t(i)\, a_{ij}\right] b_j(o_{t+1}) \qquad (2.2)$$

Etape 3 : Terminaison

$$p(O|\lambda) = \sum_{i=1}^{|S|} \alpha_T(i)$$

L'induction dans cet algorithme est fait en avant (Figure 2.2): on commence par le calcul de la probabilité $\alpha_1(i)$ du premier symbole o_1 de la séquence $\{o_1, o_2, ..., o_T\}$, et par la suite et pour chaque itération t de l'étape d'induction; on rajoute un symbole o_{t+1} telle que $t \in \{1,...,T-1\}$ et on réitère la procédure jusqu'à ce qu'on calcule la probabilité de génération de la séquence entière $O = \{o_1, o_2, ..., o_T\}$. À la fin, on déduit la probabilité $p(O|\lambda)$ en basant sur la probabilité $\alpha_T(i)$, $\forall i \in S$, d'avoir généré la séquence O et d'être arrivé à l'état i à l'instant T.

La première étape de cet algorithme nécessite une opération (Une multiplication) pour chaque état $i \in S$, donc une complexité d'ordre $O(|S|)$. Et concernant la phase d'induction, pour chaque couple (j, t), on réalise N opérations (Environ ($|S|$-1) additions et ($|S|$+1) multiplications), si on prend en considération tous les couples on aura une complexité d'ordre $O(|S|^2 T)$. Alors le calcul de la probabilité $p(O|\lambda)$ à l'aide de l'algorithme Forward requiert au total que $O(|S|^2 T)$ opérations.

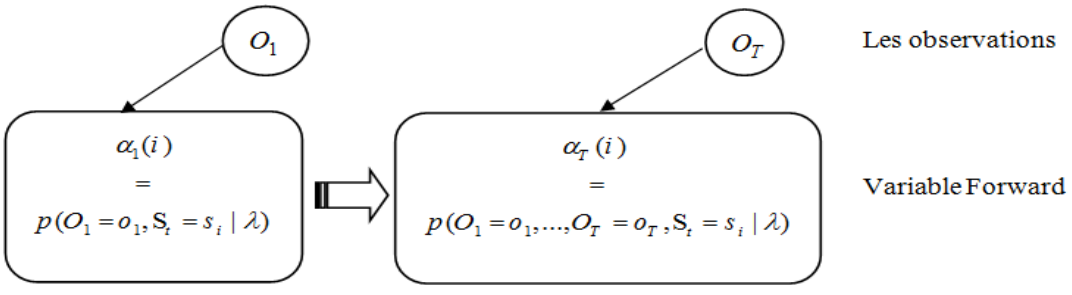

Figure 2.2- Procédure de l'algorithme Forward

L'algorithme Backward est similaire à l'algorithme Forward mais à l'envers. La variable $\beta_t(i)$ (2. 3) nommée variable de Backward, elle désigne la probabilité d'avoir généré la séquence $\{o_1, o_2, ..., o_T\}$ en partant de l'état s_i à l'instant t, étant donné le modèle λ.

$$\beta_t(i) = p(O_{t+1} = o_{t+1}, ..., O_T = o_T \mid S_t = s_i, \lambda) \ , \ t < T \tag{2.3}$$

Algorithme 2.2- Backward

***Etape* 1** : Initialisation, pour tout $i \in S$:

$$\beta_T(i) = 1$$

***Etape* 2** : Induction, pour tout $t \in [T-1, 1]$ et $i \in [1, |S|]$:

$$\beta_t(i) = \left[\sum_{j=1}^{|S|} a_{ij} \, \beta_{t+1}(j) b_j(o_{t+1}) \right] \tag{2.4}$$

L'étape d'induction pour chaque couple (i,t) de cet algorithme nécessite $(2|S|)$ multiplications et $(|S|-1)$ additions. Si on prend en considération tous les couples on aura une complexité d'ordre $O(|S|^2 T)$. Alors l'algorithme Backward requiert au total $O(|S|^2 T)$ opérations.

L'induction dans l'algorithme Backward est fait en arrière : on commence par l'initialisation de la probabilité $\beta_T(i)$ à l'instant T, et par la suite et pour chaque itération t de l'étape d'induction, on introduit le symbole o_{t+1} telle que t∈{T-1, T-2, …, 1} et on réitère la procédure jusqu'à ce qu'on calcule la probabilité de génération de la séquence entière $O = \{o_1, o_2, ..., o_T\}$.

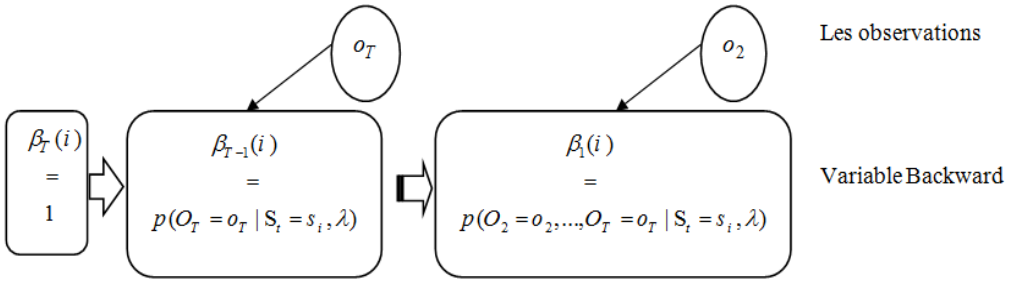

Figure 2.3- Procédure de l'algorithme Backward

Après le calcul de la variable Forward $\alpha_t(i)$ et la variable Backward $\beta_t(i)$, nous sommes en mesure de calculer la vraisemblance $p(O \mid \lambda)$ de la séquence d'observations $O = \{o_1, o_2, ..., o_t\}$, pour le modèle λ, à chaque instant t. Nous avons :

$$\alpha_t(\mathrm{i}) \times \beta_t(\mathrm{i}) = p(\mathrm{O}_1 = o_1, ..., \mathrm{O}_t = o_t, S_t = s_i \mid \lambda) \times p(\mathrm{O}_{t+1} = o_{t+1}, ..., \mathrm{O}_T = o_T \mid S_t = s_i, \lambda)$$
$$= p(\mathrm{O}_1 = o_1, ..., \mathrm{O}_T = o_T, S_t = s_i \mid \lambda)$$

En sommant le produit $\alpha_t(i) \times \beta_t(i)$ sur l'ensemble des états cachés i du MMC, alors la vraisemblance peut être exprimée par :

$$p(O \mid \lambda) = \sum_{i=1}^{|S|} \alpha_t(i) \beta_t(i) \tag{2.5}$$

2.3.2 Algorithme Viterbi

L'algorithme Viterbi (Algorithme 2.3) est inventé par l'ingénieur American Andrew James Viterbi en 1967 [100], il calcule la suite d'états cachés la plus probable vu les observations $\mathrm{O} = \{\mathrm{o}_1, \mathrm{o}_2, ..., \mathrm{o}_T\}$.

L'algorithme Viterbi permet de résoudre le problème de décodage. On définit pour chaque sommet (i , t), $\forall t \in [1, T]$ et $\forall i \in [1, |\mathrm{S}|]$, la variable de Viterbi $r_t(i)$ qui représente la probabilité maximal des chemins se terminant par s_i :

$$r_t(i) = \max_{S_1, ... S_{t-1}} p(S_1 = s_1, ..., S_t = s_i, \mathrm{O}_1 = o_1, ... \mathrm{O}_t = o_t \mid \lambda) \tag{2.6}$$

On introduit de plus une variable $\psi_1(i)$ qui mémorise le chemin localement optimal emprunté. Pour chaque sommet (j,t), on stocke dans la variable $\psi_t(j)$ l'état du MMC qui a pu obtenir la valeur maximal du produit $r_{t-1}(i) \times \mathrm{a}_{ij}$ à l'instant t-1.

Algorithme 2.3- Viterbi

Etape 1 : Initialisation, pour $i \in [1, |\mathrm{S}|]$

$$r_1(i) = \pi_i \times b_i(o_1) \text{ et } \psi_1(i) = 0$$

Etape 2 : L'étape récursive, pour $2 \le t \le T$ et $j \in [1, |\mathrm{S}|]$

$$r_t(j) = \max_{1 \le i \le |S|} r_{t-1}(i) \times b_j(o_t) \times a_{ij}$$

$$\psi_t(j) = \arg\max_{1 \le i \le |S|} r_{t-1}(i) \times a_{ij}$$

Etape 3 : Terminaison

$$s_T^* = \arg\max_{1 \le i \le |S|} \mathrm{r}_T(\mathrm{i})$$

Etape 4 : Détermination du meilleur chemin $s^* = \{s_1^*, ..., s_T^*\}$ avec $s_t^* = \psi_{t+1}(s_{t+1}^*)$, t = T-1,…, 1.

Au niveau de la première étape de l'algorithme, le calcul de la variable de Viterbi $r_t(i)$ nécessite en totalité une seule opération pour chaque état $\mathrm{i} \in \mathrm{S}$ donc elle a une complexité d'ordre $\mathrm{O}(|\mathrm{S}|)$. La phase récursive est la plus couteuse au niveau de la complexité, pour chaque couple (j,t) nous avons |S| opérations (Environ $2\times|\mathrm{S}|$ multiplications), si nous prenons en considération tous les couples nous aurons une complexité d'ordre $\mathrm{O}\left(|\mathrm{S}|^2 T\right)$. En totalité, l'algorithme Viterbi a une complexité d'ordre $\mathrm{O}\left(|\mathrm{S}|^2 T\right)$.

2.3.3 Algorithme Baum-Welch

L'algorithme le plus communément utilisé pour résoudre le problème d'apprentissage est l'algorithme de Baum-Welch il porte les noms de Leonard E. Baum et Lloyd Welch.

Etant donnés un MMC $\lambda = \{\Pi, A, B\}$ et une observation O, l'algorithme Baum-Welch (Algorithme 2.4) cherche le MMC $\lambda *$ qui a la plus forte probabilité d'engendrer la séquence O. En fait, c'est une procédure de ré-estimation itérative de la matrice de transition A, de la matrice d'observation B et de la probabilité d'initialisation Π telle que :

$$\lambda^* = \arg\max_{\lambda} = p(O = o \mid \lambda) \tag{2.7}$$

Pour faire ces ré-estimations, on fait appel aux deux variables Forward et Backward, et on définit deux nouvelles variables $\xi_t(i, j)$ et $\gamma(i, \mathrm{t})$:

- La variable $\xi_t(i, j)$ (2.8) représente la probabilité d'être dans l'état s_i à l'instant t et dans l'état s_j à l'instant t+1, étant donné l'observation O et le modèle λ :

$$\xi_t(i, j) = p(S_t = s_i, S_{t+1} = s_j \mid O, \lambda) \tag{2.8}$$

On a :

$$\xi_t(i, j) = \frac{\alpha_t(i) \times \mathrm{b}_i(o_{t+1}) \times a_{ij} \times \beta_{t+1}(j)}{\sum_{k=1}^{|\mathrm{S}|} \alpha_t(\mathrm{k})\beta_t(\mathrm{k})} \tag{2.9}$$

- La variable $\gamma(i,t)$ (2.10) représente la probabilité d'être dans l'état s_i à l'instant t étant donné l'observation O et le modèle λ :

$$\gamma(i,t) = p(S_t = s_i \mid O) \tag{2.10}$$

De la même façon on peut écrire la formule (2.10) de $\gamma(i,t)$ en utilisant les variables Forward et Backward :

$$\gamma(i, \mathrm{t}) = \frac{\alpha_t(i) \times \beta_t(\mathrm{i})}{\sum_{j=1}^{|\mathrm{S}|} \alpha_t(\mathrm{j})\beta_t(\mathrm{j})} \tag{2.11}$$

Et par la suite on peut faire la ré-estimation en suivant ces formules :

- La matrice de probabilité initiale ré-estimée :

$$\pi_1(i) = \gamma(i,1) \tag{2.12}$$

- La matrice de probabilité de transition ré-estimée :

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma(i,t)} \tag{2.13}$$

- La matrice de probabilité d'observation ou bien d'émission ré-estimés:

$$b_i(o_m) = \frac{\sum_{t=1 \cap o_t = o_m}^{T} \gamma(i,t)}{\sum_{t=1}^{T} \gamma(i,t)} \tag{2.14}$$

Après avoir ré-estimé les différents paramètres du modèle de départ λ, l'algorithme recalcule la vraisemblance du nouveau modèle λ'. Il va ensuite, réitérer les différentes étapes de ré-estimation précisées ci-dessus suivant le nouveau modèle λ', tant que la vraisemblance courante $p(O|\lambda')$ n'est pas maximale.

Algorithme 2.4- Baum-welch

Faire

Appliquer les algorithmes Forward (algorithme 2.1) et Backwad (algorithme 2.2)

Pour t=1 à T faire

Pour i=1 à |S| faire

Pour j=1 à |S| faire

Calculer $\xi_t(i,j)$

Fin pour

Calculer $\gamma(i,t)$

Fin pour

Fin pour

Ré-estimer $\lambda = \{\Pi, A, B\}$

Tant que : il y'a augmentation de $p(O|\lambda)$

2.4 Conclusion

Dans ce chapitre, nous avons présenté les **Modèles de Markov Cachés** (MMCs) qui sont des processus stochastiques. Ils sont utilisés dans une variété d'applications, mais la reconnaissance vocale et l'analyse des séquences biologiques telle que la modélisation de la séquence d'ADN restent les deux domaines d'applications les plus importants. Nous avons discuté en détail les trois problèmes de base qui sont soulevés par l'utilisation de ce genre de modèle.

3 Problèmes Décisionnels de Markov (PDMs)

Dans le chapitre précèdent, nous avons présenté d'un côté le cadre formel des Modèles de Markov Cachés (MMCs), et de l'autre, les différents algorithmes conçus pour leurs résolutions. Nous allons décrire en détail dans ce chapitre un autre type de Modèle Markovien (MM) : les **Problèmes Décisionnels de Markov** (PDMs), et plus particulièrement sur les PDMs sous le critère d'actualisation. Dans un second temps, nous étudierons la classe de complexité de ces modèles et nous présenterons deux algorithmes classiques de résolution : l'algorithme d'**Itération de la Valeur** (IV) et l'algorithme d'**Itération de la Politique** (IP).

Les progrès dans le domaine des PDMs de larges dimensions contribuent énormément à les développer et à dépasser leurs limites. Ainsi, dans la dernière section, nous allons citer des contributions de certains auteurs pour l'accélération de la convergence des solveurs des PDMs.

D'ailleurs, ces travaux prennent plusieurs directions :

- Le **développement de l'équation de récursivité** de l'algorithme IV classique ce qui a donné plusieurs extensions de ce dernier : l'algorithme d'**Itération de la Valeur Pré-Jacobi** (IV-PJ), l'algorithme d'**Itération de la Valeur Pré-Gauss-Seidel** (IV-PGS), l'algorithme d'**IV-Gauss-Seidel** (IV-GS) et l'algorithme d'**Itération de la Valeur Jacobi** (IV-J). Ces nouvelles extensions sont tous caractérisées par un temps de convergence meilleur que la version classique [17, 52, 61, 79] ;
- Approche de **sur-relaxation** successive qui représente une variante de la méthode de **Gauss Seidel** [104, 40, 88] ;
- La combinaison des algorithmes ou ce qui est connu sous le nom d'**hybridation d'algorithmes** tel que la version d'algorithme d'**Itération de la Politique Amélioré** (IP-A) [82] introduite en 1978 par Puterman et Shin ;
- **Elimination d'actions** qui consiste à éliminer l'ensemble des actions strictement ou faiblement dominées par d'autres actions [65] ;
- **Décomposition ou agrégation** de l'espace d'états en se basant sur la technique de conception d'algorithmes « Diviser pour régner » [20, 73, 89, 1, 2, 30] ;
- Approche de **partitionnement et priorisation** [101, 102, 103] d'espace d'états, elle vise à partitionner l'espace d'états en régions et par la suite on priorise une région devant l'autre en comparant la valeur de priorité calculée.

3.1 Cadre formelle des PDMs

Les PDMs [81, 19, 9] s'appuient sur la théorie mathématique des CMs, ils sont une façon naturelle de formaliser des problèmes de décision dans l'incertain sans mémoire du passé vérifiant l'hypothèse de Markov.

Considérons un système dynamique observé aux instants t = 1, 2,... (Figure 3.1). A chaque instant t l'état du processus est noté par s_t où s_t est une variable aléatoire à valeurs dans l'ensemble d'états S. Si le processus est dans l'état s, un contrôleur choisit une action appartient à l'ensemble des actions de s. L'action choisie à l'instant t peut être considérée comme une variable aléatoire a_t. Si le système est dans l'état s et une action a est choisie, alors indépendamment de l'historique du processus, il en résulte deux choses:

- Une récompense r(s,a) est acquis immédiatement ;
- Le système passe à un autre état s' avec une probabilité de transition $p(s'|s,a)$ et le processus continu de la même façon à partir de l'état s'.

Un PDM est donc composé de quatre éléments {S, A, P, R} :

- $S = \{s_1, s_2, ..., s_{|S|}\}$: est l'espace d'états fini de dimension |S| ;
- $A = \{a_1, a_1, ..., a_{|A|}\}$: est l'espace des actions qui contrôlent la dynamique de l'état ;
- $P: S \times A \times S \to [0,1] \equiv p(s'|s,a),\ \forall s,\ s' \in S,\ \forall a \in A$: la matrice des probabilités de transition entre états, $p(s'|s,a)$ représente la probabilité de passer de l'état s à l'état s' quand l'action a est exécutée ;
- $R: S \times A \to \mathbb{R}$: la fonction de récompense, $r(s,a)$ est la récompense obtenue quand on est dans l'état s et on exécute l'action a.

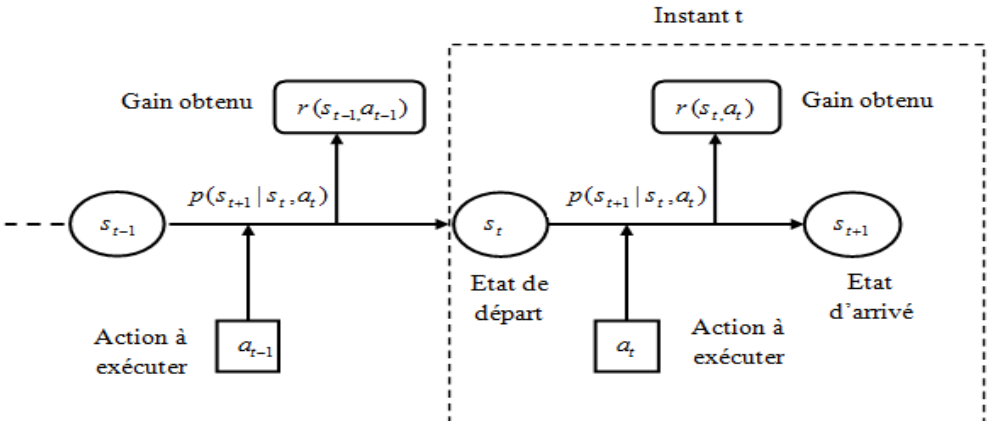

Figure 3.1- Schéma explicatif de PDM

Lorsque l'ensemble d'états S est de dimension finie, on peut représenter le PDM par un graphe G(S,U) dont les nœuds (ou bien les sommets) sont les états $s \in S$ du processus et

l'ensemble des arcs U={(s,s')$\in S \times S$:$\exists$ a$\in$ A(s) et p(s'|s,a)>0 }. Pour rendre la représentation graphique plus significative et riche d'information sur l'évolution du système; on ajoute les actions, les récompenses et les probabilités de transition correspondants à chaque état s. Afin de différencier entre les actions de chaque état s dans le graphe, on ajoute des flèches dont le nombre des flèches sur chaque arc désigne une action données $\{a_1, a_2, ..., a_{|A(s)|}\}$ comme il est illustré sur la figure 3.2.

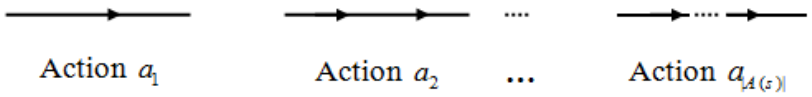

Figure 3.2- Représentation des actions dans sa forme graphique de l'état s

Par la suite, chaque arc du graphe résultant doit porter une étiquette $(r(s,a), p(s'|s,a))$ indiquant le gain et la valeur de probabilité correspondante à l'état s en exécutant l'action a. La figure 3.3, illustre un exemple de représentation graphique d'un PDM avec quatre états $S = \{s_1, s_2, s_3, s_4\}$ et un espace d'actions $A(s) = \{a_1, a_2\}$ pour tout état s$\in$S. L'état s_1 est caractérisé par :

- L'espace d'action $A(s_1) = \{a_1, a_2\}$;
- Les gains $r(s_1, a_1) = 3$ et $r(s_1, a_2) = 2$;
- Les probabilités de transition vers les autres états : $p(s_1|s_1,a_1) = 1$, $p(s_2|s_1,a_1) = 0$, $p(s_3|s_1,a_1) = 0$, $p(s_4|s_1,a_1) = 0$, $p(s_1|s_1,a_2) = 0$, $p(s_2|s_1,a_2) = \frac{1}{2}$, $p(s_3|s_1,a_2) = 0$ et $p(s_4|s_1,a_2) = \frac{1}{2}$.

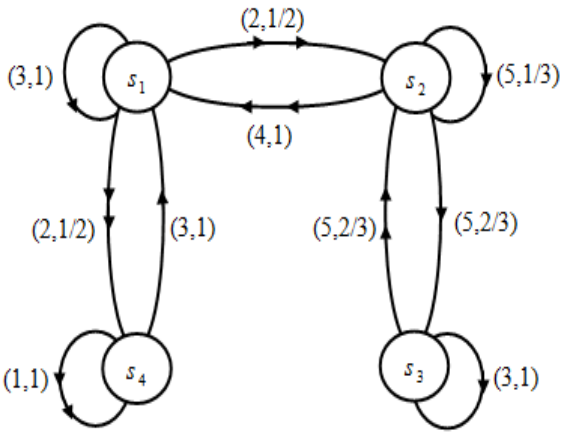

Figure 3.3- Représentation graphique d'un PDM à quatre états

Dans la théorie des PDMs, l'objectif principal est la recherche d'une règle de décision, qui précise pour un agent décideur, quelle action à choisir dans chaque état s à un instant t donné. Afin de différencier deux règles de décisions, on utilise ce qu'on appelle « Critère

d'optimalité » qui a pour objectif d'indexer les politiques qui permettront de générer les séquences de récompenses les plus importantes possibles.

3.2 Critères d'optimalité

Une politique ou bien une stratégie c'est la fonction qui définit le comportement de l'agent dans chaque situation possible. Ainsi, pour chaque état s du système, elle doit préciser une probabilité d'émission des actions A.

Une stratégie π est dite stationnaire si elle ne dépend pas du temps t ($\pi^t = \pi^{t'}$) et sa règle de décision peut être définie par :

$$\pi : \mathrm{S} \times \mathrm{A} \rightarrow [0,1] \tag{3.1}$$

$$(s, \mathrm{a}) \rightarrow \pi(\mathrm{s}, \mathrm{a}) = \mathrm{p}(\mathrm{a}_t = a \mid \mathrm{S}_t = s) \tag{3.2}$$

Une stratégie stationnaire est dite déterministe quand on associe une seule action à exécuter a pour chaque état s :

$$\pi : \mathrm{S} \rightarrow \mathrm{A} \tag{3.3}$$

$$\mathrm{s} \rightarrow \pi(\mathrm{s}) = a \tag{3.4}$$

Les critères d'optimalité se basent sur l'optimisation des récompenses possibles en utilisant l'espérance mathématique, ils permettent de définir une mesure d'utilité nommée fonction valeur V^{π} associée à chaque politique π :

$$\mathrm{V}^{\pi} : S \rightarrow \mathbb{R} \tag{3.5}$$

Une politique optimale π^* est la politique qui maximise l'espérance de l'agent en tout état s de l'espace d'état S, soit :

$$\mathrm{V}^{\pi}(s) \leq \mathrm{V}^{\pi^*}(s), \ \forall s \in S \tag{3.6}$$

Ou bien :

$$\pi^* \in \arg\max_{\pi} \mathrm{V}^{\pi} \tag{3.7}$$

Alors selon qu'on utilise la somme, la somme actualisée ou la moyenne des récompenses, on dit que la fonction valeur repose respectivement sur le critère du gain total, le critère d'actualisation (total pondéré) ou le critère de la moyenne. La différence entre ces trois critères réside dans la manière dont sont évalués les historiques. En fait, ces critères sont des instances d'utilités espérées [23] où les valeurs des historiques jouent le rôle des utilités. La similarité entre ces critères réside, d'une part, dans leur formule additive en r_t , qui résume la totalité des récompenses reçues le long d'une trajectoire avec r_t la variable aléatoire qui

désigne le gain reçu à l'instant t et, d'autre part, l'espérance E[.] qui résume la distribution des récompenses pouvant être reçues le long de la même trajectoire.

L'horizon de temps d'un PDM définit une précision temporelle au-delà de laquelle ce qui peut arriver ne nous intéresse pas, il est aussi connu par : l'ensemble des étapes de décision. L'horizon est assimilé à un sous ensemble de $\mathbb{N}$. Prenons l'exemple d'un PDM à horizon fini, soit :

$$t = \{0, 1, ..., T\},\ T \in \mathbb{N}^* \tag{3.8}$$

Les performances du système sont évaluées en considérant uniquement ses T prochaines décisions, dans le cas des PDMs à horizon infini les performances du système sont évaluées en considérant une infinité des décisions, et dans ce cas nous avons :

$$t = \mathbb{N} \tag{3.9}$$

3.2.1 Critère d'actualisation

Le critère d'actualisation, ou le critère total pondéré, est le critère à horizon infini le plus classique. Sa fonction valeur est celle qui associe à tout état s la limite lorsque le nombre des étapes de décision tend vers l'infini de l'espérance en suivant la politique π à partir de s de la somme des futures prochaines récompenses pondérées par un facteur d'actualisation γ, l'intérêt de ce dernier est d'assurer la convergence, sa valeur est comprise entre 0 et 1. La fonction valeur correspondante peut être définie par :

$$V_\gamma^\pi(s) = E^\pi[r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^t r_t + ... \mid s_0 = s],\ s \in S \tag{3.10}$$

$$V_\gamma^\pi(s) = E^\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right] \tag{3.11}$$

La formule qui vient d'être précisée montre l'intérêt de ce critère, il représente un compromis entre la récompense à court terme (conséquence immédiate de l'exécution d'une action), et la récompense qui pourra résulter de l'exécution de cette action mais à plus long terme t.

3.2.2 Critère du gain total

Le critère du gain total est similaire au critère d'actualisation sauf qu'on fixe le facteur d'actualisation γ à un. Ainsi, la fonction valeur pour ce critère peut être définie par :

$$V^\pi(s) = E^\pi[r_0 + r_1 + r_2 + ... + r_t + ... \mid s_0],\ s \in S \tag{3.12}$$

$$V^\pi(s) = E^\pi\left[\sum_{t=0}^{\infty} r_t \mid s_0 = s\right] \tag{3.13}$$

3.2.3 Critère de la moyenne

En utilisant le critère de la moyenne nous ne travaillons pas avec la somme pondérée des récompenses mais avec la moyenne des récompenses le long d'une trajectoire n. On peut définit alors la fonction valeur associée à une politique particulière π et à un état fixé s par :

$$V^{\pi}(s) = \liminf_{n\to\infty} \frac{1}{n} E^{\pi}[r_0 + r_1 + r_2 + ... + r_{n-1} \mid s_0] \tag{3.14}$$

$$V^{\pi}(s) = \liminf_{n\to\infty} E^{\pi}[\frac{1}{n}\sum_{t=0}^{n-1} r_t \mid s_0] \tag{3.15}$$

3.3 Complexité des Problèmes Décisionnels de Markov

Les PDMs de grandes tailles est l'un des axes de cette thèse, alors la réaction de ces modèles face à l'augmentation du nombre de données à traiter est primordiale, cette réaction est expliquée en générale par les classes de complexité que nous avons introduites dans le premier chapitre. En 1987, Papadimitriou et Tsitsiklis [72] ont présenté le théorème suivant qui donne la classe de complexité des PDMs sous différents critères d'optimalité.

Théorème 3.1 : le problème des PDMs est dans la classe P-complet dans les trois cas : horizon fini, critère de la moyenne et critère d'actualisation.

L'inclusion de ce modèle dans la classe P-complet provoque une résolution en temps polynomial pour une machine de Turing déterministe. Ainsi, le problème peut être décidé en temps polynomial demandant un temps d'exécution constant pendant la résolution.

3.4 Algorithmes classiques de résolution des PDMs

La programmation dynamique est une méthode générale de résolution des problèmes d'optimisations séquentielles à temps discrètes. En particulier, elle définit le cadre général de deux méthodes classiques de résolution des PDMs à savoir : l'algorithme IV [36, 75] ; et l'algorithme IP [53, 52], autres que la programmation linéaire [31, 70]. On va présenter dans les deux paragraphes suivants le principe des deux premiers algorithmes pour un PDM sous le critère d'actualisation.

3.4.1 Algorithme d'Itération de la Valeur

L'algorithme IV est l'un des algorithmes standards de résolution des problèmes modélisés par des PDMs. Il se décline en deux versions utilisées respectivement au cas à horizon fini et au cas à horizon infini, nous allons concentrer dans ce travail sur la deuxième version. Cet algorithme détermine une stratégie ε-optimale dans le cas des espaces d'états et d'actions finis.

L'algorithme 3.1 commence par une valeur (Fonction valeur) nulle $V^{0}(s) = 0$ pour chaque état s∈S, puis il calcule itérativement la valeur de chaque état jusqu'à une condition d'arrêt.

La valeur d'un état à l'itération t est calculée à partir de sa valeur à l'itération t-1**.** En pratique, plusieurs conditions d'arrêt peuvent être envisagées, la plus classique consiste à stopper l'itération lorsque $|V^t(s)-V^{t-1}(s)|<\varepsilon, \forall s \in S$, où ε est un seuil d'erreur fixé à priori. La fonction valeur obtenue est alors sous-optimale, plus ε est proche de 0 plus la valeur des états est proche de la valeur optimale. Le choix de la politique est fait en gardant pour chaque état l'action qui a pu donner la valeur maximale.

Algorithme 3.1- Itération de la Valeur

t=0
Pour tout $s \in S$
  $V^t(s)=0$
Fin pour
Répéter
  t ← t+1
  Pour tout $s \in S$ faire

$$V^t(s)=\max_a\left[r(s,a)+\gamma\sum_{s'\in S}P(s'|s,a)V^{t-1}(s')\right] \quad (3.16)$$

$$\pi^t(s)=\arg\max_a\left[r(s,a)+\gamma\sum_{s'\in S}P(s'|s,a)V^{t-1}(s')\right] \quad (3.17)$$

  Fin pour
Jusqu'à $\max_s |V^t(s)-V^{t-1}(s)| < \varepsilon$
Retourner π^t

Comme la complexité est une approche indépendante des caractéristiques de la machine ou du langage utilisé, elle présente un calibre très pertinent pour mesurer la performance des algorithmes. L'algorithme IV en termes d'efficacité dépend de deux contraintes: la complexité d'une itération et le nombre d'itérations nécessaire pour converger qui est généralement difficile à déterminer. Dans la partie itérative, pour chaque couple (état i, action a) on réalise O(|S|) opérations, environ de |S| additions et |S| multiplications. Si nous prenons en considération tous les couples dans le système, nous aurions une complexité d'ordre $|A||S|^2$ pour chaque itération.

3.4.2 Algorithme d'Itération de la Politique

L'algorithme IP est un algorithme plus complexe que l'algorithme IV. Il nécessite l'initialisation d'une première politique arbitraire, que l'algorithme va ensuite l'améliorer par itérations successives. On peut subdiviser l'algorithme en deux phases :

- La phase d'évaluation : elle revient à résoudre un système de |S| équations à |S| inconnues (3.20).

- La phase d'amélioration : elle consiste à améliorer cette politique courante en déterminant pour chaque état une nouvelle action qui améliore l'ancienne valeur (3.21).

A chaque politique π_k , on associe la Chaîne de Markov de matrice de transition $\mathrm{P}(\pi_k)$ et le vecteur des récompenses $r(\pi_k)$ définies par :

$$\left[P(\pi_k)\right]_{ss'} = \sum_{a \in A(s)} P(s' \mid s,a)\pi_k(s,a) \; ; \; s, s' \in S \tag{3.18}$$

$$\left[r(\pi_k)\right]_s = \sum_{a \in A(s)} r(s,a)\pi_k(s,a) \tag{3.19}$$

Dans une première étape (Voir l'algorithme 3.2), on résout directement l'équation de Bellman (3.20) ce qui revient à résoudre le système d'équations linéaires avec |S| équations à |S| inconnues :

$$\mathrm{V}_{\pi_k}{}^{\gamma}(s) = \left[r(\pi_k)\right]_s + \gamma\left[P(\pi_k)\right]_{ss'} \mathrm{V}_{\pi_k}{}^{\gamma}(s)$$

$$[I - \gamma\left[P(\pi_k)\right]_{ss'}]\mathrm{V}_{\pi_k}{}^{\gamma}(s) = \left[r(\pi_k)\right]_s$$

$$\mathrm{V}_{\pi_k}{}^{\gamma}(s) = [I - \gamma\left[P(\pi_k)\right]_{ss'}]^{-1}\left[r(\pi_k)\right]_s \tag{3.20}$$

Puis, dans un second temps, on améliore la politique courante en vérifiant l'inéquation :

$$r(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a)\, \mathrm{V}_{\pi_k}(s') > \mathrm{V}_{\pi_k}(s), \forall s, s' \in S \text{ et } a \in A(s) \tag{3.21}$$

En se basant sur le principe que si deux politiques successives sont identiques alors elles sont nécessairement optimales. Ainsi, l'algorithme s'arrête lorsque $\pi_k = \pi_k'$.

Algorithme 3.2- Itération de la Politique

$k \leftarrow 0$
Initialisation aléatoire de la politique π_k
Répéter
  $\pi_k \leftarrow \pi_k'$
  Pour $s \in S$ faire
    Calculer $\mathrm{V}_{\pi_k}(s)$ en résolvant le système (3.20)
  Fin pour
  Pour chaque $s \in S$ faire
    si $\exists a \in A(s)$: $r(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a)\, \mathrm{V}_{\pi_k}(s') > \mathrm{V}_{\pi_k}(s)$ alors $\pi_k'(s) \leftarrow a$
    Sinon $\pi_k'(s) \leftarrow \pi_k(s)$
    Fin si

$k \leftarrow k+1$
Jusqu'à $\pi_k = \pi_k^{'}$
Retourner V_k, π_k

Contrairement à l'algorithme IV qui nécessite un grand nombre d'itérations pour converger et chaque itération est très rapide, cet algorithme n'exige qu'un nombre faible d'itérations pour converger, mais chaque itération est très coûteuse. Chaque itération de l'algorithme consiste en deux opérations : la résolution du système d'équations, qui nécessite $|S|^3$ opérations, et la phase d'amélioration, qui est effectuée en $|A|\ |S|^2$ opérations.

3.5 Problèmes Décisionnels de Markov de grandes tailles

L'utilisation des PDMs comme cadre de modélisation et de résolution des problèmes réels, aboutit souvent à des espaces d'états multidimensionnels et donc de très grandes tailles. Cette contrainte sur la taille limite la possibilité de résolution numérique et donc la portée pratique des PDMs. Les efforts des chercheurs pour résoudre cette problématique vont être détaillé dans ce qui suit.

3.5.1 Développement de l'équation de récursivité

Comme la précision et la rapidité pendant la recherche de la fonction valeur exige des efforts considérables de calcul, le développement de l'équation de récursivité qui permet d'accélérer les algorithmes fondamentaux a été un objectif principal de la recherche. Ces méthodes sont connues en anglais sous le nom «Value Itération (VI) Schemes », tels que :

- L'algorithme d'IV-PJ [17, 52] qui représente la forme la plus simple de l'algorithme classique IV :

$$V^n(s) = \max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a)\, V^{n-1}(s') \right], \forall s \in S, n = 1,2,... \qquad (3.22)$$

- L'algorithme d'IV-PGS a été suggéré par Kushner [61], Porteus [79] et Reetz [86]. C'est une version améliorée proposée par Gauss-Seidel, son avantage c'est l'utilisation de la mise à jour de la fonction valeur avant que l'itération arrive à sa fin, alors que la version de IV-PJ attend l'itération suivante pour qu'elle utilise la dernière mise à jour.

$$V^n(s) = \max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' < s} p(s' \mid s,a)\, V^{n}(s') + \gamma \sum_{s' \geq s} p(s' \mid s,a)\, V^{n-1}(s') \right], \forall s \in S \quad (3.23)$$

- L'algorithme d'IV-J et l'algorithme d'IV-GS sont des améliorations respectives des algorithmes IV-PJ et IV-PGS [79, 52]. Pour tout état $s \in S$, les équations de récursivité peuvent être données respectivement sous les formes suivantes :

$$V^{n}(s) = \max_{a \in A(s)} \left\{ \left(r(s,a) + \gamma \sum_{\substack{s' \in S \\ s' \neq s}} \mathrm{p}(s' \mid \mathrm{s}, \mathrm{a})\, \mathrm{V}^{n-1}(\mathrm{s}') \right) / \left(1 - \gamma p(\mathrm{s} \mid \mathrm{s}, \mathrm{a})\right) \right\} \tag{3.24}$$

$$V^{n}(s) = \max_{a \in A(s)} \frac{r(s,a) + \gamma \sum_{s'<s} \mathrm{p}(s' \mid \mathrm{s}, \mathrm{a})\, \mathrm{V}^{n}(\mathrm{s}') + \gamma \sum_{s'>s} \mathrm{p}(s' \mid \mathrm{s}, \mathrm{a})\, \mathrm{V}^{n-1}(\mathrm{s}')}{1 - \gamma p(\mathrm{s} \mid \mathrm{s}, \mathrm{a})} \tag{3.25}$$

3.5.2 Approche de sur-relaxation successive

Ce n'est qu'une variante de la méthode de Gauss-Seidel pour la résolution des systèmes linéaires, elle est inventée par Young en 1950 [104], afin de résoudre des systèmes linéaires de façon automatique avec des ordinateurs. Après le travail de Young, le domaine de la recherche a connu la naissance de plusieurs extensions de cette méthode telles que les travaux de Frankel [40] et Riley [88]. La méthode de sur-relaxation est déjà utilisée auparavant en 1946 par Lewis Fry Richardson et R. V. Southwel mais seulement pour l'utilisation humaine.

L'idée principale de cette approche est de faire la mise à jour de la fonction valeur V^{n} (3.26) autrement; en remplaçant la fonction V^{n-1} par le terme $\underline{V}$ qui représente une combinaison linéaire de la fonction valeur V^{n-1} à l'instant n-1 et la fonction valeur V^{n-2} à l'instant n-2 :

$$\underline{V} = \omega\, V^{n-1}(s) + (1-\omega)\, V^{n-2}(s) \tag{3.26}$$

Le coefficient ω est le coefficient de relaxation, il a une valeur dans l'intervalle]1,2[, plusieurs travaux faites pour préciser la meilleure valeur de ce dernier afin d'avoir une accélération de la convergence, Kushner et Kleinman [61] ont testé l'accélération de la convergence en fixant le facteur de relaxation sur des PDMs non actualisés.

3.5.3 Hybridation d'algorithmes

La technique d'hybridation est une sorte de combinaison d'algorithmes pour arriver à un nouvel algorithme. En 1978 Puterman et Shin [82] ont introduit une nouvelle version d'algorithme IP nommée IP-A qui combine les caractéristiques de l'algorithme IV et l'algorithme IP. Dans cette version, l'évaluation de la politique se fait par une itération sur les valeurs $V_{\pi_k}(s)$, s∈S, pour une politique fixée π_k. On commence (Algorithme 3.3) par l'affectation d'une valeur nulle $V_{\pi_k}^{t}(s) = 0$ à chaque état s∈S, puis on calcule successivement les valeurs de chaque état en suivant l'équation 3.27 sous la politique d'entrée π_k jusqu'à une condition d'arrêt.

Algorithme 3.3- Itération de la Politique Amélioré
Seuil de précision ε

$k \leftarrow 0$
Initialisation aléatoire de la politique π_k
Répéter
t $\leftarrow 0$
Pour chaque $s \in S$ faire : $V^{t}_{\pi_k}(s) = 0$
Répéter
Pour chaque $s \in S$ faire

$$V^{t+1}_{\pi_k} = r(s, \pi_k(s)) + \gamma \sum_{s' \in S} P(s' \mid s, \pi_k(s))\, V^{t}_{\pi_k}(s') \tag{3.27}$$

Fin pour
t $\leftarrow t+1$
Jusqu'à $| V^{t}_{\pi_k}(s) - V^{t-1}_{\pi_k}(s) | < \varepsilon$
Pour chaque $s \in S$ faire

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a)\, V^{t}_{\pi_k}(s') \right] \tag{3.28}$$

Fin pour
k $\leftarrow$ k+1
Jusqu'à $\pi_k = \pi_{k-1}$

Cette version reste plus efficace que la version classique de l'algorithme IP. La complexité de cet algorithme peut être résumée dans la complexité du calcul des deux équations (3.27) et (3.28) qui sont respectivement d'ordre $|S|^2$ et $A|S|$ pour chaque itération.

3.5.4 Elimination d'actions

La dominance des actions ou bien la dominance d'une stratégie représente l'un des concepts récemment utilisés, cherchant à réduire l'ensemble des actions en éliminant les actions ou bien les stratégies k strictement dominantes (3.29) ou faiblement dominantes (3.30).

$$V^{t}_{k}(s) < V^{t}_{d}(s), \forall d \in A(s), s \in S \tag{3.29}$$

$$V^{t}_{k}(s) \leq V^{t}_{d}(s), \forall d \in A(s), s \in S \tag{3.30}$$

Logiquement ça serais inutile de choisir une stratégie k strictement dominée par d'autres stratégies d. D'après le théorème de Porter (2008) [78], la complexité de déterminer si une stratégie donnée peut être éliminée est P complet s'il s'agit d'une élimination des stratégies strictement dominées et NP-complet s'il s'agit d'une élimination des stratégies faiblement dominées.

MacQueen est le premier à avoir inventé le concept d'élimination d'actions en 1967 [65] sur des PDMs à espace d'états fini, il a proposé un simple test (3.31) qui autorise l'élimination d'une action k :

$$V_k^t(s) < V^t(s) + \gamma(a_t - b_t)/(1-\gamma) \tag{3.31}$$

Avec :

$$a_t = \min_s \left(V^t(s) - V^{t-1}(s)\right) \text{ et } b_t = \max_s \left(V^t(s) - V^{t-1}(s)\right)$$

Après quatre ans Porteus a modifié le test en 1971 [80], tel que $\forall d \in A(s)$ on a:

$$V_k^t(s) < V_d^t(s) + \gamma^2(a_{t-1} - b_{t-1})/(1-\gamma) \tag{3.32}$$

3.5.5 Décomposition et Agrégation

Comme la taille des données limite l'utilisation pratique des MMs et surtout des PDMs, plusieurs travaux se sont focalisés sur la notion d'espace d'état. Ces travaux sont inspirés de la technique « Diviser pour régner », nous pouvons les partager en deux grandes catégories: des méthodes à base d'agrégation d'états qui visent à regrouper les états du système par caractéristiques communes ; et des méthodes de décomposition des espaces d'états et d'actions.

Diviser pour régner provient du latin « Divide and conquer», dans le domaine d'informatique, c'est une technique de conception d'algorithme réduisant récursivement un problème de départ de grande taille en un ou plusieurs sous-problèmes de tailles plus petites. Il existe plusieurs algorithmes qui relèvent de ce paradigme [43]: la recherche par dichotomie dans un tableau, la tri-fusion, la multiplication de Karatsuba et l'exponentiation rapide ou la recherche des plus proches points dans un plan. L'un des algorithmes très classiques qui utilise cette méthode astucieuse c'est la recherche dichotomique dans un tableau, elle représente un processus itératif ou récursif de recherche où, à chaque étape, on coupe en deux parties (pas forcément égales) un espace de recherche qui devient restreint à l'une de ces deux parties.

Soit un tableau T de dimension N déjà trié par ordre croissant, que les éléments du tableau sont comparables. Le problème posé consiste à rechercher un élément k dans le tableau T et, s'il est présent, retourner son indice dans le tableau. La première version de l'algorithme (algorithme 3.4) est la version naïve puisqu'il consiste à parcourir le tableau à partir du premier élément, et d'arrêter dès que l'on trouve l'élément k.

Algorithme 3.4- Rechercher classique

```
i = 0
Tant que i < (N-1) faire
    Si T[i] = k alors retourner i
    Sinon retourner NonTrouver
    i=i+1
```

L'utilisation de cette version, nous oblige à parcourir tout le tableau ce qui implique une complexité linéaire d'ordre O(N). Prenons l'exemple du tableau T=23, 50, 84, 100, 137, 170, 200, en recherchant la valeur k=137 dans notre tableau T en suivant le chemin coloré dans la figure 3.4.

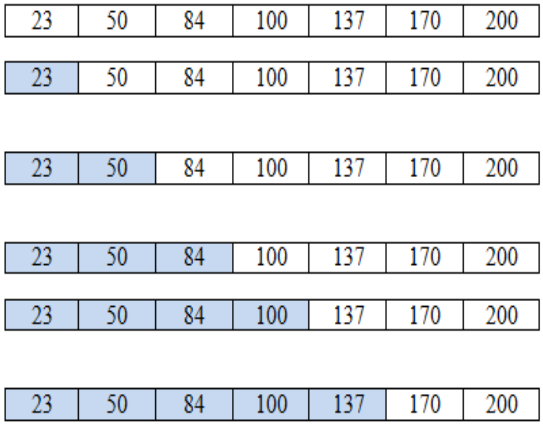

Figure 3.4- Exemple de recherche classique dans un tableau

Et pour la recherche dichotomique (Algorithme 3.5), nous commençons par comparer le nombre k au nombre qui se trouve au milieu M du tableau T. Si c'est le même, nous avons trouvé le k, sinon nous recommençons sur la première moitié dont l'indice est inférieur à M (ou la seconde dont l'indice est supérieur à M) selon la valeur cherchée est ce qu'elle est plus petite (ou plus grande) que le nombre qui se trouve au milieu M du tableau.

Algorithme 3.5- Recherche dichotomique

$i = 1 \; ; \; j = N \; ; \; M = \left[\frac{(i + j)}{2} \right]$

Tant que $\left(k \neq T\,[i\,] \wedge (i < j)\right)$ faire

Si $\left(k \neq T\,[M\,] \wedge (i < j)\right)$ alors j=M-1

Sinon i=M+1

M=$\left[\frac{(i + j)}{2} \right]$

$Si\,(k == T\,[i\,])$ alors retourner M

Sinon retourner -1

Reprenons l'exemple ci-dessus en utilisant dans la recherche l'algorithme 3.5.

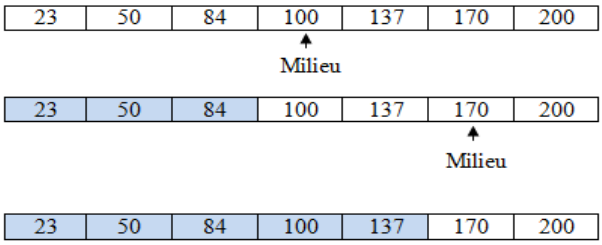

Figure 3.5- Exemple de recherche dichotomique

L'utilisation de la recherche dichotomique réduit la complexité d'un ordre linéaire à une complexité logarithmique O(log(N)).

- Technique de décomposition

La complexité de la plus part des algorithmes de résolution des problèmes modélisés par des MMs est polynomiale en termes du nombre d'états du modèle. Ainsi, l'utilisation de ces algorithmes dans leur formalisme classique est rédhibitoire pour des problèmes de grandes tailles tels que les MMCs et les PDMs. L'approche de la décomposition elle-même est divisé en deux catégories différentes suivant le type du MM étudié: les méthodes de décomposition en série [30, 73] dont la tâche globale de l'agent ou bien du décideur est vue comme un ensemble de tâches séquentielles; contrairement aux méthodes de décomposition parallèle [15 , 48, 66, 93, 94] dont la tache globale est vue comme un ensemble de tâches concurrentes.

- Les méthodes sérielles : elles sont utilisées lorsque l'on peut subdiviser l'espace d'états en union de sous-espaces d'états:

$$S = \{S_1, S_2, .., S_n\} \tag{3.33}$$

  Ces sous-espaces associés doivent être exécutés l'un après l'autre.

- Les méthodes parallèles : elles sont utilisées lorsque l'espace d'états du modèle peut être vu comme un produit cartésien des sous-espaces d'états :

$$S = \{S_1 \times S_2 \times .. \times S_n\} \tag{3.34}$$

Dans ce dernier cas, on est simultanément devant plusieurs décideurs qui cherchent soit des objectifs distincts ou bien le même objectif mais de différentes manières et on doit prendre en considération leurs interactions.

Dans cette thèse, nous allons concentrer nos efforts essentiellement sur le type séquentiel de décomposition. Nous pouvons préciser, dans ce sens, les travaux les plus connus de Ross et Varadarajan [89], T Dean [30], Abbad et Boustique [1] et Abbad et Daoui [2]. Ils utilisent des algorithmes qui calculent des stratégies optimales sous les différents critères précisés auparavant (Le critère d'actualisation, le critère du gain total et le critère de la moyenne) contrairement aux autres solutions proposées dans la littérature et qui sont adaptées seulement à certains critères et qui ne donnent que des solutions approximatives. L'idée maîtresse de cette décomposition est d'agréger les états en sous modèles (Ou bien en régions ou des sous

espaces) suivant des caractéristiques communes de façon à subdiviser la charge du calcul des politiques. Ainsi, elle permet d'accélérer le calcul en construisant une hiérarchie entre des problèmes locaux et une solution globale. Les sous-espaces d'états des sous modèles doivent être plus petits que l'espace d'états du modèle global.

La décomposition de Ross et Varadarajan est une technique inspirée du travail de Bather en 1973 [7] et un peu généralisée par Kallenberg en 2002 [58]. Considérons le critère de la moyenne, les auteurs construisent des modèles restreints à des classes fortement communicantes et un ensemble des états transitoires T. La résolution de ces sous modèles passe en deux étapes : on commence par la résolution des PDMs retreints à chaque classe fortement communicante en utilisant l'algorithme de programmation linéaire, par la suite on agrège ces PDMs restreints en états et on résout le PDM agrégé associé. Ensuite, on génère une solution optimale du problème initial à partir des solutions du problème agrégé.

La décomposition proposée par Abbad et Boustique pour les PDMs sous le critère de la moyenne est une technique inspirée du travail d'Avsar et al.. Elle se base sur le même principe mais en déterminant que les classes communicantes qui représentent les « Composantes Fortement Connexes » (CFCs detaillé dans le chapitre 5) pour le graphe associé au PDM. Ces classes sont ensuite réparties dans des niveaux (PDMs restreints correspondants à chaque niveau, voir figure 3.6).

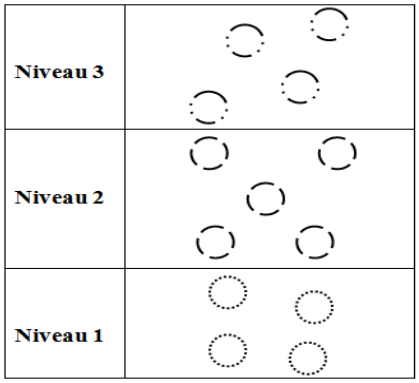

Figure 3.6- Classement des cercles en niveaux suivant la forme

Le travail proposé par Abbad et Daoui a pour objectif le balayage des autres critères, ils ont traité les PDMs sous le critère d'actualisation et le critère pondéré.

- Technique d'agrégation :

Quand on est devant la modélisation et la résolution d'un système qui représente une collection de sous-systèmes plus ou moins indépendants, on est devant une extension du PDM nommé PDM factorisé ou bien PDM à espace d'états factorisé qui est présentée en 1995 par Boutilier [20]. Il est possible de prendre en compte la relative indépendance de ces sous-systèmes pour décrire plus efficacement le modèle du problème afin de faciliter la recherche de la solution, ce qui est connu sous le nom d'agrégation. Cette technique permet de

représenter les fonctions de transition et de récompense d'un problème de façon compacte, en exploitant la structure du problème.

Dans cette période Boutilier a créé un nouvel axe de recherche suivi par plusieurs chercheurs [21, 47, 94] dont l'objectif est d'identifier des représentations convenables au dynamique d'évolution de cette extension et d'une autre part, d'inventer des algorithmes de planification de stratégies d'actions optimales ou sous-optimales.

3.5.6 Partitionnement et priorisation des états

Les algorithmes classiques restent des algorithmes conceptuellement simples et faciles à mettre en œuvre, mais en raison de leurs inefficacités en temps et en espace dans le cas des modèles de larges dimensions, ils sont incapables de produire de bonnes solutions. Wingate et Seppi en 2003, 2004 et en 2005 [101, 102, 103] ont créé une autre direction de recherche pour débloquer cette problématique. Ils ont introduit deux autres notions : le partitionnement et la priorisation pour éviter le calcul des taches redondantes ou inutiles.

La priorité nous permet d'identifier à partir de l'espace du problème les régions à maximal productivité ce qui est connu sous le nom du priorisation, on calcule une valeur de priorité pour chaque état s à une instant t en se basant sur la fonction d'erreur de Bellman:

$$\max_{a \in A} \left\{ r(\mathrm{s},\mathrm{a}) + \gamma \sum_{s'} P(s' \mid s, a) V^{t}(s') \right\} - V^{t}(s) \qquad (3.35)$$

La priorité d'une partition est définie comme étant la priorité maximale d'un état s au sein de la partition, tel que la dépendance entre les partitions est liée aux dépendances entre les états constituant chaque partition. Soient p et p' deux partitions, on dit que p dépend de p' s'il existe s et s' avec $s \in p$, $s' \in p'$ tel que :

$$P(s' \mid s, a) \neq 0$$

Par la suite les partitions seront traitées par ordre de priorité en sélectionnant la plus haute priorité p, on itère tous les états $s \in p$ jusqu'à la convergence. À la fin on recalcule la priorité de tous les partitions p' dépendantes de p, on procède de la même façon jusqu'on termine.

Ultérieurement plusieurs travaux [4, 56, 106] ont terminé le développement et l'application de cette technique dans plusieurs domaines et elle a montré un grand succès en termes de temps de convergence.

3.6 Conclusion

Dans ce chapitre, nous avons détaillé le cadre théorique des Problèmes Décisionnels de Markov (PDMs) ainsi que leur **complexité**. Nous avons, en particulier, décrit l'algorithme **d'Itération de la Valeur** (IV) et l'algorithme **d'Itération de la Politique** (IP), qui représentent deux méthodes classiques de résolution des PDMs. A la fin, nous sommes

revenues plus en détail sur les **PDMs de grandes tailles** et les différents travaux réalisés en littérature pour alléger la problématique de la dimensionnalité. Le chapitre suivant se focalisera sur la technique de **parallélisation** des PDMs comme une autre méthode de développement.

4 Parallélisation des Problèmes Décisionnels de Markov

Après avoir introduit dans les parties précédentes le contexte général et les concepts clefs de notre travail. Dans ce chapitre, nous allons aborder le concept du **parallélisme** et son utilisation dans les Problèmes Décisionnels de Markov (PDMs). A partir des années 60, le besoin de puissance de calcul dans plusieurs domaines, n'a fait qu'augmenter, ce qui a donné la naissance à des architectures parallèles. Ceci a aussi donné la naissance à des modèles de programmation, des langages et des bibliothèques parallèles qui ont pour objectif d'approcher le concept du parallélisme aux programmeurs.

Dans un premier temps, nous allons décrire la différence entre le calcul séquentiel et le calcul parallèle. Nous allons introduire par la suite, les différents **types d'architectures** parallèles créées par **Flynn** et la modification faite sur cette classification par Duncan, puis nous allons montrer quelques métriques qui permettent d'évaluer les performances des solutions parallèles. Finalement, nous discuterons l'application du parallélisme au niveau du PDM.

4.1 Calcul séquentiel et parallèle

Le calcul séquentiel (Figure 4.1) est basé sur la subdivision du problème en une série d'instructions discrètes qui sont exécutées séquentiellement l'un après l'autre et à tout moment dans le temps une instruction sera exécutée sur un seul processeur.

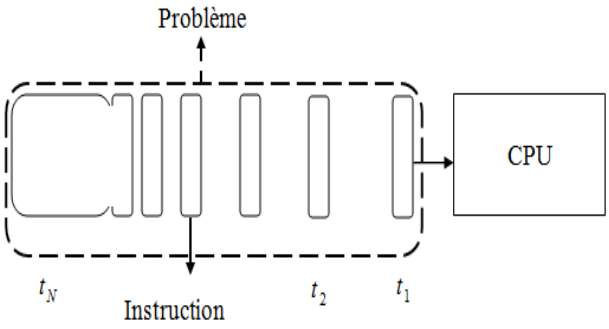

Figure 4.1- Calcul séquentiel du problème en un seul processeur

Le calcul parallèle (Figure 4.2) est différent du calcul séquentiel, plusieurs opérations peuvent être exécutées simultanément. Initialement, le problème est divisé en parties distinctes qui peuvent être résolus simultanément et on le décompose aussi en une série d'instructions qui seront exécutées simultanément sur plusieurs processeurs.

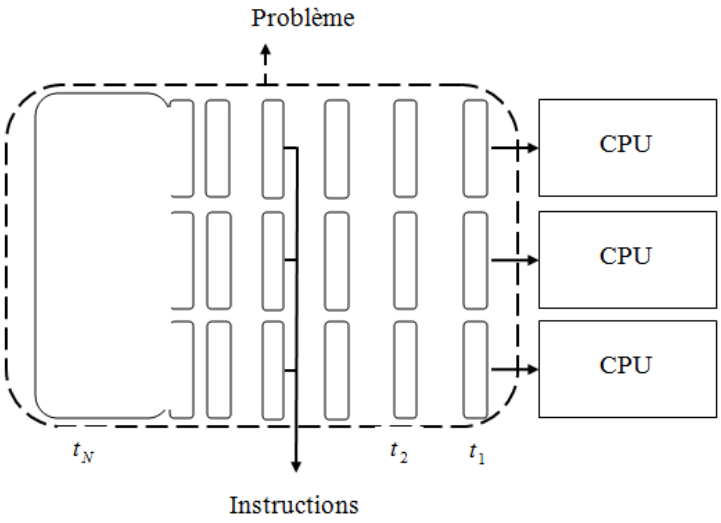

Figure 4.2- Calcul parallèle sur plusieurs processeurs

4.2 Taxonomie de Flynn

Le scientifique américain Flynn [37, 38] dans les années 60 a classé les architectures parallèles en quatre grandes catégories (Figure 4.3) : SISD, SIMD, MISD et MIMD en se basant sur les notions de flot de contrôle « **I**nstruction » et flot de données « **D**ata ». Cette classification n'est pas parfaite, et certaines architectures ne rentrent pas vraiment dans les catégories de la classification de Flynn.

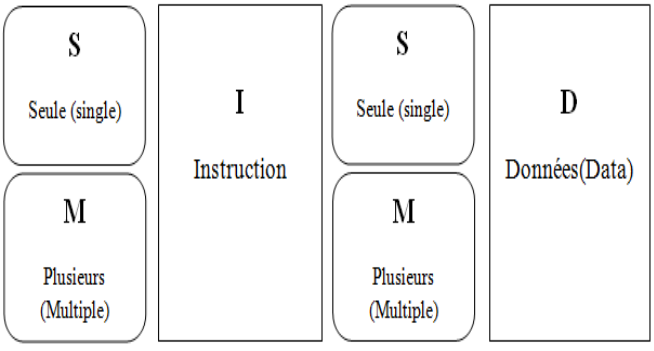

Figure 4.3- Classification des architectures parallèles suivant Flynn

- Machine SISD (**S**ingle **I**nstruction **S**ingle **D**ata) (Figure 4.4) : Dans ce type de machines, à chaque moment donné une seule instruction est exécutée et une seule

donnée est traitée, c'est une machine séquentielle incapable de faire toute forme de parallélisme, ce que l'on appelle d'habitude une machine de « Von Neuman ».

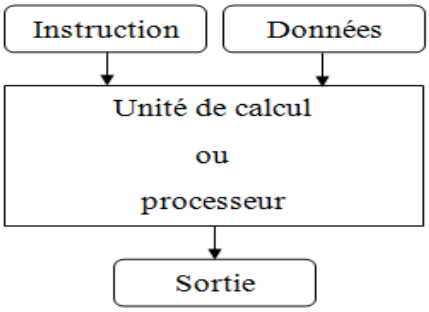

Figure 4.4- Machine SIMD

- Machine SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata) (Figure 4.5) : le parallélisme des données est la caractéristique majeure de ce genre de machines où chaque processeur exécute la même instruction sur ses propres données locales. Elles ont été plus ou moins abandonnées en raison de leur manque de souplesse sauf pour le domaine de traitement d'image. Les machines CM-2 [98] et MPP [6] sont des représentants les plus connus de cette catégorie.

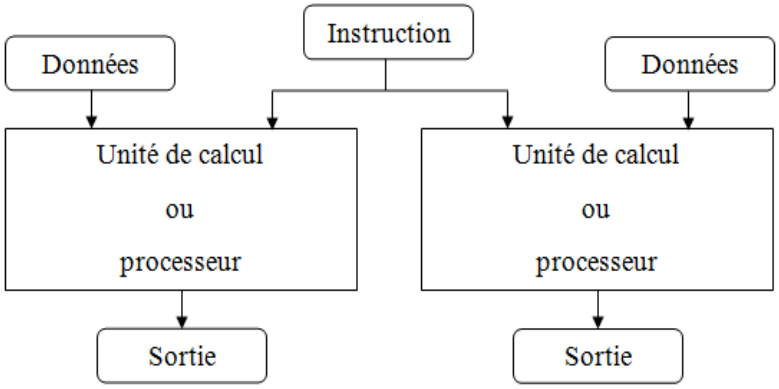

Figure 4.5- Machine SIMD

- Machine MISD (**M**ultiple **I**nstruction **S**ingle **D**ata) (Figure 4.6) : c'est la catégorie d'architectures la plus rare, elle peut exécuter des instructions différentes en parallèle sur une donnée identique. L'exemple le plus claire de cette catégorie est les architectures systoliques et cellulaires qui sont souvent utilisées pour des applications particulières, ces architectures contiennent des cellules dont chaque cellule reçoit des données en provenance des cellules voisines et chaque cellule effectue un calcul spécial.

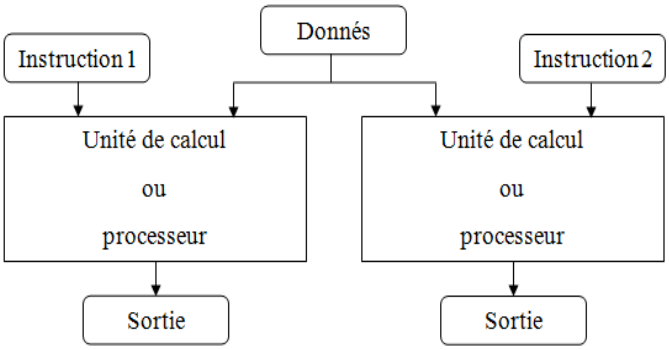

Figure 4.6- Machine MISD

- Machine MIMD (**M**ultiple **I**nstruction **M**ultiple **D**ata) (Figure 4.7) : c'est l'architecture parallèle la plus étudiée et implémentée, dont les processeurs effectuent simultanément des instructions différentes sur des données différentes. En pratique, ce n'est pas nécessaire d'utiliser des instructions différentes sur des données différentes, on exécute le même code et on parle du modèle SPMD (**S**igle **P**rogram **M**ultiple **D**ata).

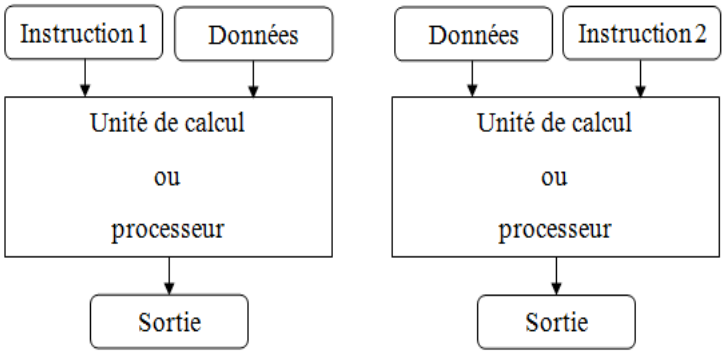

Figure 4.7- Machine MIMD

4.3 Organisation de la mémoire

De ces quatre modèles d'architectures, les modèles MIMD et SIMD sont prépondérants et correspondent à la notion du parallélisme. Par la suite, Duncan [33] a introduit des modifications à la « Taxonomie de Flynn » dans le cas où on utilise différentes données pour une ou plusieurs instructions, Duncan a proposé une nouvelle classification selon les différentes méthodes d'accès physique à la mémoire : modèle à mémoire partagée et modèle à mémoire distribuée. D'un point de vue logiciel, ces modèles sont implémentés dans des « Interfaces de Programmation Applicatives » (**A**pplications **P**rogramming **I**nterface (API)).

4.3.1 Modèle à mémoire partagée

Le modèle à mémoire partagée [29, 92] propose un espace mémoire commun accessible depuis tous les processeurs. Il permet de garder un espace d'adressage identique ce qui facilite la communication. Par contre, ce modèle ne garantit pas la cohérence des différentes copies des données en mémoire devant l'augmentation du nombre de processeurs.

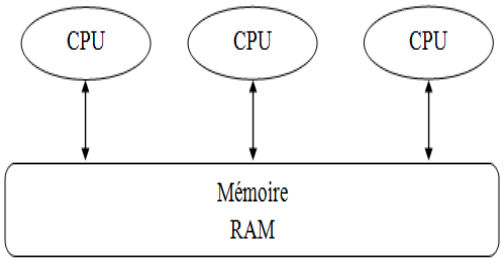

Figure 4.8- Architecture à mémoire partagée

L'OpenMP (Open **M**ulti-**P**rocessing) est imposé comme un standard de la programmation en mémoire partagée, c'est une API permettant la programmation d'applications parallèles à mémoire partagée écrites en C, C++ ou Fortran en y ajoutant des directives de compilation.

4.3.2 Modèle à mémoire distribuée

À l'inverse du modèle à mémoire partagée [13], le modèle à mémoire distribuée est constituée de plusieurs processeurs ayant chacun sa propre mémoire centrale isolée des autres. Ils sont reliés entre eux à travers un réseau d'interconnexion. Ainsi les communications doivent se faire explicitement par le programmeur en partageant des messages entre processeurs, entrainant un effort de développement supplémentaire, alors on est devant une communication de type « passage de messages ».

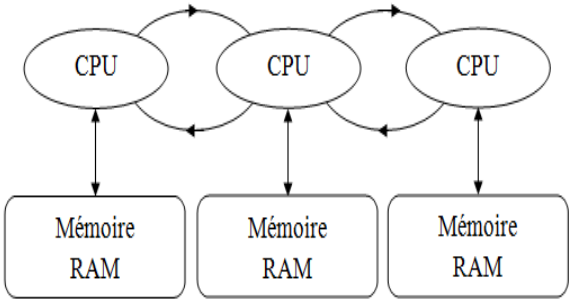

Figure 4.9- Architecture à mémoire distribuée

Le MPI (**M**essage **P**assing **I**nterface) [41, 45, 46] est l'environnement de programmation le plus connu pour les architectures à mémoires distribuées inventée en 1992 mais la version finale qui est étendue au Fortran et au C++ date de 1997. Ainsi, le MPI a supplanté toutes les autres librairies de programmation à mémoire distribuée. Les développeurs ont optimisé plusieurs versions gratuites à ce jour (OpenMPI, MPICH, MVAPICH, IBM MPI, etc.).

Le programme parallèle en utilisant le MPI, est exécuté en lançant plusieurs processus ou chaque processus gère sa propre mémoire sans connaissance de celle des autres. La communication entre ces processus, soit pour transférer des données ou bien de se synchroniser; se fait via un mécanisme d'échange de message. Elle dépend du nombre de processus participants dans la communication, nous avons deux types de communication:

- « Communication point-à-point » : c'est une communication entre deux processus, où l'un envoie un message et l'autre le reçoit.
- « Communications collectives » : c'est une communication entre plusieurs processus en un seul appel.

Pour bénéficier des avantages des deux modèles distribué et partagé, les chercheurs ont créés un modèle hybride qui combine les deux.

4.4 Performance du parallélisme

Comme si l'objectif principal du parallélisme est d'accélérer nos algorithmes, il parait logique que l'augmentation du nombre des processeurs tournés dans le programme augmente la performance du parallélisme et d'un autre coté il accélère le programme. Mais malheureusement ce n'est pas toujours le cas, l'algorithme résultant peut être moins performant que le séquentiel ou bien l'implémentation parallèle dégrade la performance. Il s'agit d'une limite théorique car les processeurs doivent communiquer entre eux et se synchroniser afin de fournir un bon résultat. Plus que le temps d'exécution tiré d'après l'exécution, il y'a le facteur d'accélération et d'efficacité qui représentent deux facteurs caractérisant le gain lié à la parallélisation.

Le facteur d'accélération A(N) (4.1) c'est le rapport du temps d'exécution du programme séquentielle t_1 au temps d'exécution du même programme parallélisé t_N :

$$A(N) = \frac{t_1}{t_N} \tag{4.1}$$

Avec N= {1, 2, 3,..} le nombre des processeurs utilisés pendant l'exécution, à partir de N=2, le facteur d'accélération A(N) doit être supérieur strictement à 1 pour dire que le programme parallèle est plus performant.

Le facteur d'efficacité E(N) (4.2) représente le rapport entre le facteur d'accélération A(N) et le nombre des processeurs N :

$$E(N) = \frac{A(N)}{N}, \ \forall \ \text{N}=\{1, 2, 3,...\} \tag{4.2}$$

Plus que la valeur d'efficacité est proche de 1, plus que le programme est plus performant.

4.5 Application du parallélisme aux PDMs

La parallélisation d'un PDM revient à trouver les valeurs et les stratégies optimales de façon parallèle, en distribuant le calcul sur plusieurs processeurs ce qui permet de réaliser le plus grand nombre d'opérations en un temps minimum. Le concept du parallélisation au niveau des PDM prend deux catégories : la première c'est la parallélisation par itération qui exprime la façon la plus naturelle et, la deuxième catégorie est basée sur la combinaison du

parallélisme et des méthodes inventées pour accélérer ce modèle. Ce qui donne une organisation des données appropriées pour le parallélisme tel que la méthode de décomposition parallèle et la méthode de priorisation et partitionnement de l'espace d'état que nous avons précisées dans le deuxième chapitre.

Les algorithmes utilisés dans la résolution des PDMs sont partagés de façon naturelle en des itérations, chaque itération prend beaucoup de temps à cause du grand traitement à effectuer. Ainsi, les chercheurs ont pensé au parallélisation par itération [107], qui revient à paralléliser le calcul au niveau de chaque itération en subdivisant l'ensemble des états S (4.3) en P groupes dans la moyenne :

$$S = \{S_1 \cup S_2 \cup ... \cup S_P\} \quad (4.3)$$

On attribue chaque groupe à un processeur. À une itération donnée t, tous les processeurs maintiennent la valeur optimale et la politique optimale obtenus dans l'itération t-1 de tous les états des autres groupes, soit en utilisant l'un des types de communication entre processeurs qu'on a précisé au paragraphe précédent, soit on fait référence à une mémoire partagée. Par la suite, chaque processeur calcule les nouvelles valeurs pour les états du groupe qui lui correspond. La figure 4.10 résume les différentes étapes de parallélisation par itération.

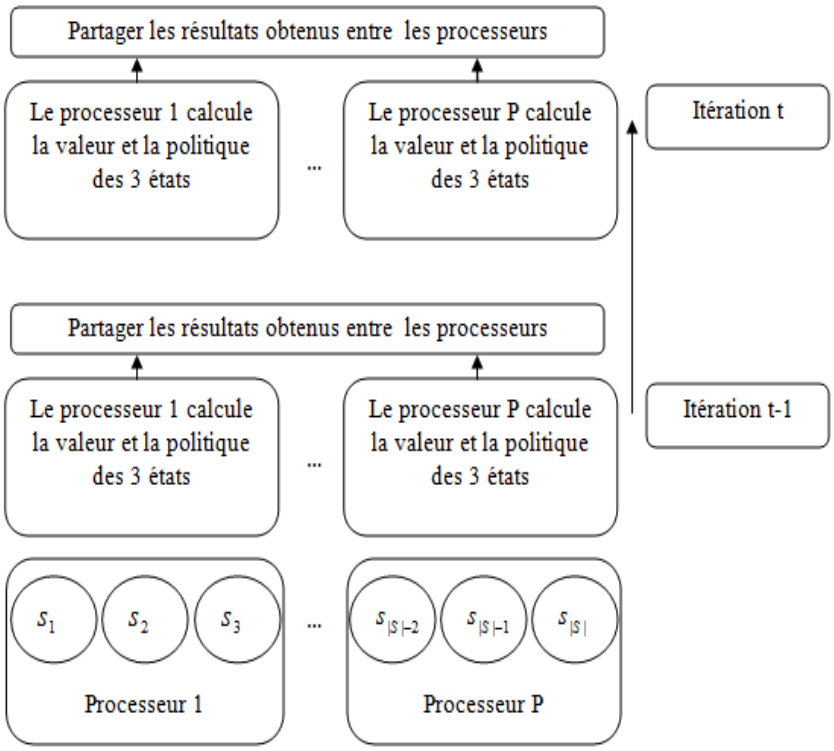

Figure 4.10- Parallélisation par itération

L'autre type de parallélisation se base sur une amélioration préalable du PDM ou bien de l'algorithme voulu.

Après avoir introduit la technique du priorisation et du partitionnement par Wingate et Seppi, ils ont pensé à accélérer plus les PDMs et plus précisément l'algorithme d'IV, en combinant ces deux techniques avec la technique du parallélisme ce qui a donné la naissance à un nouveau algorithme nommé P3VI « A Partitioned, Prioritized, Parallel Value Iterator », dont chaque technique des trois joue un rôle complémentaire. Après avoir partitionné les états en des ensembles, de telle sorte que les états qui existent dans le même ensemble sont des états mutuellement dépendants ou bien sont les moins dépendants avec les états des autres ensembles. On affecte par la suite les ensembles aux processeurs, notant que le nombre des processeurs est inférieur ou égale au nombre total des ensembles. On attribue pour chaque processeur des informations sur leur dépendance par rapport aux autres processeurs et un fil de priorité des partitions local. Eventuellement, chaque processeur va exécuter l'algorithme d'IV basé sur la technique de priorité et il informe les autres processeurs qui lui dépendent s'il y'a des changements.

La décomposition parallèle des PDMs représente une parallélisation naturelle. Particulièrement, cette technique est très utile surtout dans des systèmes multi-agents où une telle décomposition se traduit naturellement à une décomposition en agents distincts [59], chaque agent exécute une sous-tâche indépendante de l'autre agent pour arriver à atteindre l'objectif de la tâche globale de façon parallèle.

4.6 Conclusion

Dans ce chapitre, nous avons présenté le concept général du **parallélisme** ainsi que les différentes **architectures parallèles** et les librairies les plus utilisées dans ce domaine telles que le **MPI** (**M**essage **P**assing **I**nterface) et l'**OpenMP** (**Open M**ulti-**P**rocessing). A la fin, nous nous sommes intéressés plus à l'applicabilité de ce concept pour un Problème Décisionnel de Markov (PDM) afin d'accélérer ses algorithmes de résolution.

Conclusion de la partie I

Cette première partie a été consacrée aux fondements de cette thèse. Dans un premier temps, on a introduit les **Modèles Markoviens**. Par la suite, nous avons particulièrement discuté le concept de la **complexité** comme étant un calibre très pertinent pour mesurer l'efficacité des algorithmes de résolution de ces modèles. Plus tard, nous avons en particulier mis en évidence la problématique de la **multi-dimensionnalité des données** qui exprime un facteur principale limitant considérablement l'utilisation de ces modèles.

Le deuxième chapitre a porté sur une étude détaillée des **Modèles de Markov Cachés** comme étant l'un des Modèles Markoviens les plus connus, nous avons passé en revue ces différentes problématiques et nous nous sommes ensuite intéressés à leurs résolutions.

Les fondements de la formalisation des **Problèmes Décisionnels de Markov** a été l'objectif du troisième chapitre, nous avons présenté leur cadre théorique, les **critères d'optimalité** qui régissent la sélection des politiques, leur classe de complexité, ainsi que les principaux algorithmes de résolution: **Itération de la Valeur** et **Itération de la Politique** sous le critère d'actualisation. Les algorithmes classiques semblent les plus appropriés en générale, nous avons souligné leurs inefficacités dans le cas des Problèmes Décisionnels de Markov de **grandes tailles**. Dans la dernière partie de ce chapitre, nous avons précisé les différentes directions prises en littérature pour atténuer la tâche des grands calculs.

Le **parallélisme** est la nouvelle révolution dans la manière avec laquelle nous écrivons les programmes, il représente la meilleure solution quand nous ne pouvons pas avoir des résultats satisfaisants à des problèmes complexes et nécessitant une grande puissance de calcul. Vu l'importance de ce concept, on lui a réservé le dernier chapitre, nous avons détaillé l'idée conceptuelle du parallélisme et les différents classements de ces **architectures** existantes, et nous avons éclairci plus son importance par son application dans les Problèmes Décisionnels de Markov.

Cette première partie nous a permis de concevoir les grands axes directeurs de ce travail. Nous allons ainsi nous orienter vers l'introduction et la validation des approches proposées dans la deuxième partie.

Partie II : Contributions

Introduction de la partie II

Après avoir introduit dans la partie précédente le contexte général, les concepts basiques ainsi que la problématique dans lequel s'inscrit notre travail : la résolution des Modèles Markoviens de grandes tailles et plus précisément les **Problèmes Décisionnels de Markov** et les **Modèles de Markov Cachés**. Nous allons à présent, proposer des algorithmes améliorés applicable dans le cas des problèmes de grandes tailles, modélisés par l'un de ces deux modèles.

Dans le premier chapitre de cette partie, nous présenterons la **technique de décomposition** proposée par Abbad, Boustique et Daoui qui vise à décomposer le PDM initial en petits problèmes, les résoudre séparément et générer ensuite une solution du problème initial, cette technique est applicable que pour les algorithmes classiques tels que l'algorithme d'**Itération de la Valeur** et l'algorithme d'**Itération de la Politique**. Pour bénéficier de l'efficacité de cette technique nous ferons des modifications dans l'étape de construction des modèles restreints et dans la stratégie général de résolution dans la méthode de décomposition adoptée pour l'appliquer sur les Modèles de Markov Cachés dans le deuxième chapitre, et sur une méthode d'accélération de convergence de type Gauss-Seidel dans le troisième chapitre ce qui représente une approche d'hybridation entre la décomposition et une méthode d'accélération de convergence Gauss-Seidel. Nous exposerons par la suite les **algorithmes hiérarchiques** résultants ainsi qu'une étude de complexité permettant de valoriser l'application de nos approches sur les Problèmes Décisionnels de Markov et sur les Modèles de Markov Cachés. Le dernier chapitre de cette partie, débutera par l'introduction d'un procédé de **parallélisation** hybridé avec la technique de décomposition adoptée, ce genre d'hybridation nous a permis de construire des **algorithmes Hiérarchiques Parallèles** qui ne sont pas parallélisable tels que l'algorithme IV-PGS. Par la suite, une autre technique de décomposition sera développée, dite **décomposition topologique**, de même elle sera combinée avec le paradigme du parallélisme, ce qui a permis de concevoir des **algorithmes Topologiques Parallèles**.

5 Décomposition des Problèmes Décisionnels de Markov

La modélisation de la plus part des problèmes réels dans le cadre des Modèle Markoviens (MMs) reste difficile à traiter, leurs espaces sont souvent de très grandes tailles. Toutefois, comme nous l'avons déjà montré dans le chapitre 2, la totalité des états du système peuvent être représentés de façon plus concise et le problème global est décomposé en sous-problèmes relativement indépendants et, qui seront résolus puis recombinés pour avoir une solution du problème initial. Dans ce chapitre, nous allons nous focaliser sur **les techniques de décomposition** et plus précisément sur la **décomposition séquentielle** présentée par **Abbad et Daoui** [27, 28], nous allons montrer le rapport consistant qui existe entre cette décomposition et la **théorie des graphes**. Nous allons ensuite présenter ces deux étapes fondamentales à savoir la **classification en niveaux** et la construction des **problèmes restreints**. Nous concluons par les détailles des **algorithmes hiérarchiques** résultants.

5.1 Représentation graphique

Nous considérons la décomposition séquentielle proposée par Abbad, Boustique et Daoui, précisée au chapitre 2, elle se base principalement sur la théorie des graphes [68]. L'utilisation de la théorie des graphes, nous amène une interface qui facilite la modélisation des problèmes contenant des variables interagissant, et d'autre part nous apporte un formalisme pour représenter une distribution de probabilités sur l'ensemble des variables aléatoires sous forme d'un graphe.

Le stockage de la probabilité jointe P(X) d'un ensemble de variables aléatoires $X = \{X_1, X_2, ..., X_N\}$ garde toutes les données liant les variables même s'il n'existe pas une dépendance entre un certain nombre de variables, cette caractéristique provoque des places de stockage supplémentaires. Ainsi, nous pouvons voir l'importance de la représentation graphique qui nous apporte une représentation économique des fonctions de probabilité jointe connexes. Dans ce sens, la théorie des graphes exploite telles dépendances entre variables pour décomposer une large distribution en un ensemble de distributions locales beaucoup plus petites, en enrichissant leur bibliothèque par des différentes techniques de décomposition : la décomposition en arbre, la décomposition en niveaux, la décomposition en CFCs, etc. La méthode de décomposition inventée dans [1, 2, 27, 28] se base sur le même principe à fin de dépasser la problématique de la taille des données pour les PDMs.

Soit G=(X,U) le graphe associé à un MM avec $X = \{x_1, x_2, ..., x_N\}$ l'ensemble des nœuds qui représente l'espace des états du modèle et U représente l'ensemble des arcs qui sont les couples d'états (i, j) tels que la probabilité de transition de i vers j est non nulle.

Dans l'exemple ci-dessous figure 5.1, le graphe G=(X,U) c'est le graphe associé au MM définit par l'espace des états S={A,B,C,D,E,F,G,H,I,J,K} et l'ensemble des arcs U={(A,B), (B,A), (C,B), (B,C), (K,J), (J,K), (J,B), (G,B), (I,G), (G,I), (H,G), (G,H), (G,D), (D,F), (F,D), (E,D), (D,E)}.

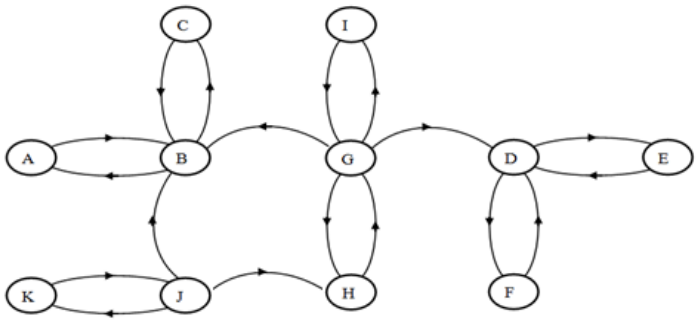

Figure 5.1- Représentation graphique d'un MM à 11 états

La méthode de décomposition considérée dans ce chapitre passe par deux grandes étapes : la première réside à décomposer par récurrence l'ensemble des états constituant le modèle en quelques niveaux sous des contraintes précises, et la deuxième étape porte sur la construction des modèles restreints à chaque niveau. Les sections suivantes passent en revue ces deux étapes.

5.2 Décomposition en niveaux

La première étape de l'approche de décomposition proposée dans [1, 2, 27, 28] est inspirée de la théorie des graphes, elle utilise plus précisément les mêmes notions pour recherche des graphes sans circuit : « L'hiérarchie des sommets » et « L'hiérarchie des niveaux » [55, 90].

La hiérarchie des sommets d'un graphe est une classification des sommets du graphe en régions numérotées de 0 à (L-1), chaque sommet x du graphe est inclus dans une région indexée par le numéro h(x) de tel sorte : l'index de la région du sommet x_1 est inférieur strictement à l'index de la région du sommet x_2 s'il existe un chemin de x_1 vers x_2 :

$$h(x_1) < h(x_2) \Leftrightarrow \text{ Il existe un chemin de } x_1 \text{ vers } x_2 \qquad (5.1)$$

Cette classification ne donne aucune région vide :

$$\text{Si } 0 \le z \le (L-1) \text{ alors } \exists\, x \in X : h(x) = z \qquad (5.2)$$

Selon le graphe de la figure 5.2, On a $X = \{x_1, x_2, x_3, x_4, x_5\}$ et U={ $(x_1,x_2),(x_1,x_5),(x_4,x_2),(x_5,x_2)$ $(x_3,x_5),(x_5,x_4)$}.

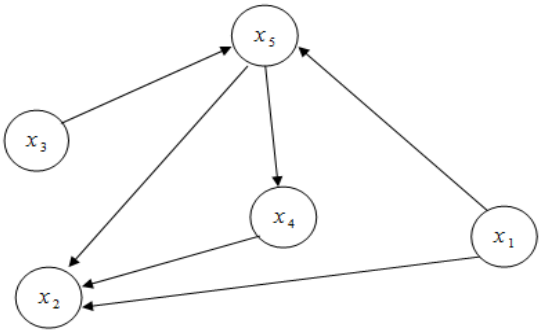

Figure 5.2- Exemple d'un graphe orienté d'ordre 5

L'index de la région du sommet x_1 est inférieur strictement à l'index des régions contenant les sommets x_5, x_4 et x_2 (Figure 5.3). On a $h(x_1) < h(x_5) < h(x_4) < h(x_2)$.

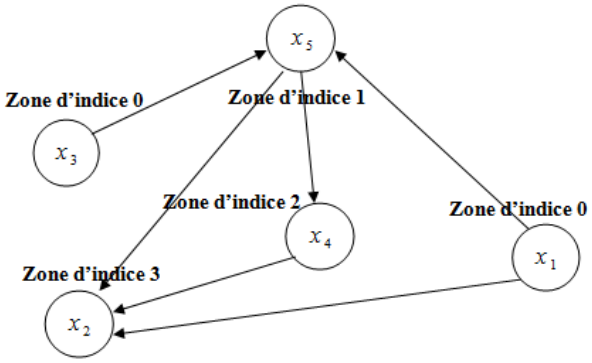

Figure 5.3- Hiérarchie des sommets

Remarque 5.1 : nous pouvons interpréter l'équation (5.1) autrement, on note que l'index de la région du sommet x_1 est égale à l'index de la région du sommet x_2 s'il n'existe pas un chemin de x_1 vers x_2 et de x_2 vers x_1 :

$$h(x_1) = h(x_2) \Leftrightarrow \text{S'il n'existe pas un chemin de } x_1 \text{ vers } x_2 \text{ et de } x_2 \text{ vers } x_1 \qquad (5.3)$$

Ce qui signifié que les sommets de la même région ne communiquent plus entre eux.

L'hiérarchie des niveaux représente une autre hiérarchisation mais plus précise. Elle s'accompagne d'une vision « topologique » ou bien d'un ordre topologique en vérifiant les deux conditions :

- Le niveau l=0 combine tous les sommets de degré intérieur nul ;
- Le niveau l={1, 2, …, (L-1)} contient les sommets dont les prédécesseurs sont dans les niveaux inférieurs à l.

On note respectivement $d^+(x)$ ($d^-(x)$) le degré extérieur (intérieur) du sommet x, c'est-à-dire le nombre d'arcs ayant x comme extrémité initiale (extrémité finale). Prenons l'exemple

montré dans la figure 5.2, le sommet x_1 est de degré intérieur nul contrairement au sommet x_2 :

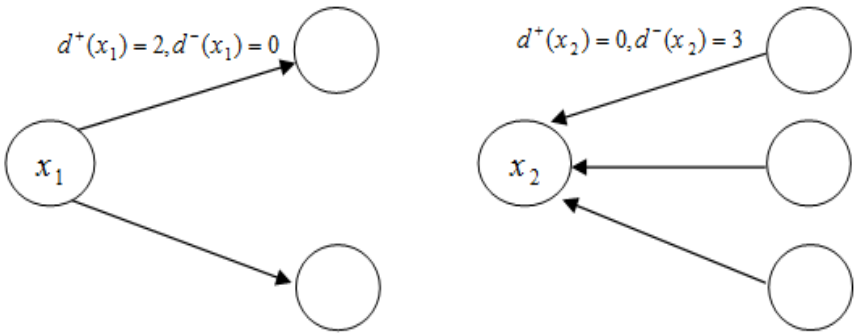

Figure 5.4- Degré intérieur et extérieur des deux sommets

On construit au début, l'ensemble des sommets N_0 sans prédécesseurs dans X et qui représente les sommets incluent dans le niveau 0. Puis, on construit par récurrence l'ensemble N_k des sommets sans précédent dans $X_k = X \setminus \{N_0 \cup ... \cup N_{k-2} \cup N_{k-1}\}$ avec k={1,2,…, (L-1)}, l'ensemble N_k forme l'ensemble des sommets inclus dans le niveau k. Comme on commence à numéroter les niveaux par zéro, le numéro du dernier niveau coïncide avec la longueur du plus long chemin du graphe.

Nous considérons l'exemple ci-dessus (Figure 5.2), nous voyons que la totalité des sommets sont répartis en quatre niveaux l={0, 1, 2 ,3}:

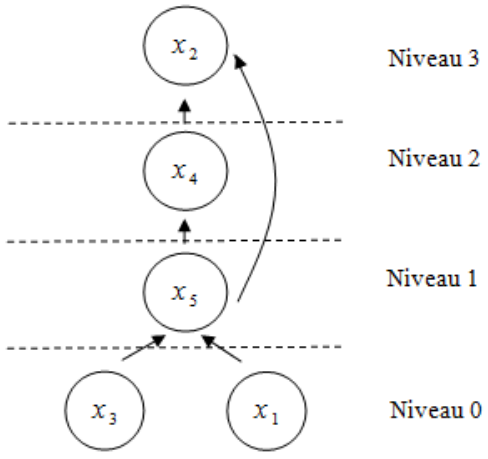

Figure 5.5- Hiérarchie des niveaux

Remarque 5.2 : On note que si on a arrivé à classer l'ensemble des sommets X en niveaux l={0, 1, …,(L-1)}, on remarque que si un sommet x appartient à un niveau $k < (L-1)$, il existe un chemin de x vers un certain y qui se trouve dans un niveau supérieur strictement à k.

Comme il est montré dans l'exemple la dessus, à partir du sommet x_5 nous pouvons arriver aux niveaux 2 et 3 grâce à un chemin respectivement vers x_4 et x_2.

L'approche de décomposition adoptée a tiré profit de ses deux types d'hiérarchisation mais avec quelques modifications, afin de prendre en considération les caractéristiques du MM étudié. Généralement, la décomposition passe par deux étapes : la première c'est une hiérarchisation en classes fortement connexes au lieu d'une hiérarchisation en sommets ou en niveaux, ce qui est connu en littérature par la décomposition en niveaux. Cette étape lui-même peut être partagée en deux parties la répartition de l'espace d'états en CFCs et la classification de ces classes dans des niveaux. La deuxième étape consiste à la construction des petits problèmes ou problèmes restreints aux classes de chaque niveau.

5.2.1 Hiérarchie en Composantes Fortement Connexes

Dans cette étape, on commence par partitionner l'espace d'état S en CFCs, $C_1, C_2, ..., C_H$, à partir de ces classes on construit par récurrence les niveaux $L_0, L_1, ..., L_{n-1}$. La recherche des CFCs peut être faite de deux manières, soit en se basant sur des traitements et une analyse précises sur la matrice d'adjacence [42] ce qui est connu sous le nom de la « fermeture transitive », ou bien sur le principe de la recherche en profondeur qui reste la meilleur façon pour le faire au niveau de la complexité.

En théorie des graphes la décomposition en CFCs d'un graphe G(X, U) orienté revient à tirer des sous-graphes de G possédant la propriété suivante: pour tout couple $(i, j) \subset U$ dans ce sous-graphe, il existe un chemin de i vers j, ou i=j, ces deux sommets sont donc en relation d'équivalence: réflexive, symétrique et transitive [24] .

Reprenons l'exemple illustré dans la figure 5.1, la décomposition du graphe G en quatre sous graphes peut être donnée par la figure 5.6.

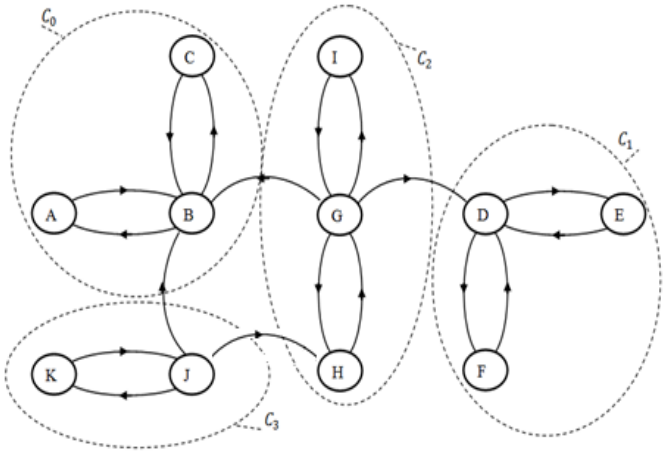

Figure 5.6-Décomposition en CFCs

Ce dernier graphe comporte quatre CFCs : C_0={A,B,C}, C_1={F,D,E}, C_2={I,G,H}, $C_3 = \{K, J\}$.

Les algorithmes de recherche des CFCs les plus connus sont : l'algorithme des puissances, l'algorithme Roy Warshall, l'algorithme de Tarjan et l'algorithme de Kosaraju.

- ***Fermeture transitive***

Afin de vérifier la relation de transitivité entre les sommets montrés dans la définition des CFCs, il doit exister un chemin entre tous les sommets appartenant au même classe, ce qui représente l'objectif principale de la fermeture transitive [44, 50, 97]. On pourra bien entendu déterminer, de manière évidente, les CFCs en trouvant la fermeture transitive $G^+(X, U^+)$ du graphe $G(X, U)$ avec X l'ensemble des nœuds de dimension n et U^+ l'ensemble des arcs. La fermeture transitive permet de déterminer pour tout couple de sommets x_i et x_j s'il existe un chemin reliant le premier au second, elle se calcule à partir des matrices binaires telles que la matrice d'adjacence.

Considérons le graphe G(X, U) (Figure 5.7), dont on va calculer sa fermeture transitive dans ce qui suit.

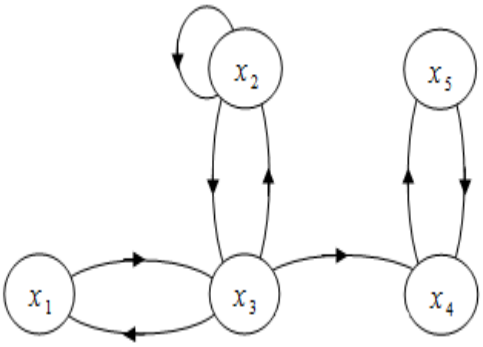

Figure 5.7- Exemple de graphe pour le calcul de la fermeture transitive

La matrice d'adjacence M_G correspondante au graphe G(X, U), est une matrice carrée d'ordre n avec $M_G(x_i, x_j) = 1$ si le couple $(x_i, x_j) \in U$, et $M_G(x_i, x_j) = 0$ sinon.

$$M_G = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Soit $G_0(X, U_0)$ (Figure 5.8) le graphe initial, complété d'une boucle pour chaque sommet de G(X, U), ce qui implique que pour tout sommet $x_{i=1,2,..,5}$ on a $M_0(x_i, x_i) = 1$.

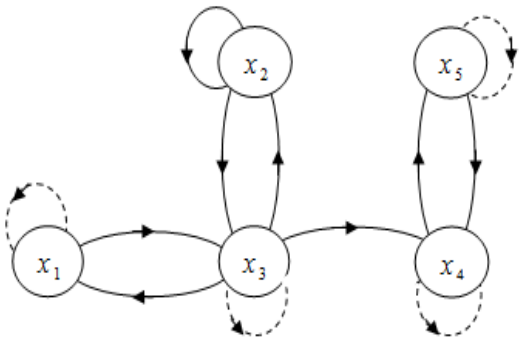

Figure 5.8- Graphe initial plus les boucles associées à ces états

Par conséquent, la matrice d'adjacence correspondante au graphe initial $G_0(X, U_0)$ est :

$$M_0 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

On note par le sous graphe $G_k(X, U_k)$ de G, k={1, 2,…, (n-1)}, définit par : $(s_i, s_j) \in U_k$ si et seulement si il existe dans G_k un chemin du sommet s_i vers s_j dont les sommets internes sont dans $\{s_1, s_2, ..., s_k\}$, on obtient alors :

$$M_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix},$$

$$M_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \text{ et } M_4 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Le chemin maximal qui relie deux sommets est le nombre de sommets moins un. Dans l'exemple traité c'est quatre, ainsi la matrice M_4 donne l'information complète sur la totalité

des chemins maximaux qui existent et même les chemins moindres, donc la matrice M_4 c'est la matrice de la fermeture transitive, nous avons :

$$M_4 = M_{G^+} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Les lignes ou bien les colonnes qui se ressemblent construisent les éléments du CFC. Dans la littérature, il existe plusieurs algorithmes pour calculer la fermeture transitive et qui visent à ajouter tous les arcs qui n'existent pas dans la matrice d'adjacence initiale du graphe G selon la règle de transitivité.

L'algorithme des puissances c'est l'algorithme le plus simple pour la recherche de la fermeture transitive, en faisant la somme booléenne des puissances successives de la matrice d'adjacence sommets-sommets et la matrice unité M_G^0. On s'arrête à la matrice de puissance n-1 :

$$M_{G^+} = \sum_{k=0}^{n-1} M_G^k \tag{5.4}$$

La matrice d'incidence résultante récapitule tous les chemins existant entre toute couple de sommets (x_i, x_j), plus les boucles réflexives (x_i, x_i) et les chemins rajoutés pour la fermeture transitive.

L'algorithme de Roy Warshall [39] (Algorithme 5.1) s'agit de l'algorithme de fermeture le plus utilisé en pratique, il repose sur le même principe que la construction fait par les puissances successives sauf qu'il est plus optimisé en exploitant la redondance induite par les transitivités, cette redondance réside dans l'existence de plusieurs chemins qui relient des sommets quelconques en différentes longueurs .

Algorithme 5.1- Roy Warshall

Soit G=(X,U) et |X|=n

Pour tout k=0 $\rightarrow$ n -1

Pour chaque sommet $x_i \in X$

Pour chaque sommet $x_j \in X$

Si $((x_i, x_j) \notin U)$ et $((x_i, x_{k+1})$ et $(x_{k+1}, x_j) \in U)$

$U = U \cup \{(x_i, x_j)\}$

Fin si
Fin pour
Fin pour
Fin pour

La construction de la fermeture transitive à l'aide de l'algorithme de puissance et de l'algorithme de Roy-Warshall a une complexité respectivement $O(n^4)$ et $O(n^3)$. Bien que Roy-Warshall reste l'algorithme le plus répondu pour calculer la fermeture transitive mais devant une complexité pareille il reste inapplicable pour le calcul des CFCs.

- ***Parcours en profondeur***

Le parcours en profondeur (depth-first) permet d'appliquer une recherche en profondeur en descendant vers les arcs les plus profonds dans le graphe en revenant à chaque fois au dernier sommet pour lequel il reste encore des successeurs à visiter.

Algorithme de Tarjan [96] inventé par l'informaticien américain Robert en 1972, basé sur le parcours en profondeur dans une seule passe, il prend en entrée un graphe orienté ou bien un digraphe et renvoie une partition des sommets du graphe correspondants à ses CFCs, c'est un algorithme de recherche qui progresse à partir d'un sommet x_i en s'appelant récursivement sur chaque sommet voisin de x_i. On débute (Algorithme 5.2) par un sommet quelconque, on l'empile dans une pile. Si le sommet de la pile a des voisins qui ne sont pas dans la pile, alors on sélectionne l'un de ces voisins et on l'empile et en le marquant de son numéro de découverte, sinon on le dépile de la pile. On recommence cette procédure tant que la pile n'est pas vide.

Nous allons nous appuyer sur le graphe d'exemple cité dans la figure 5.7

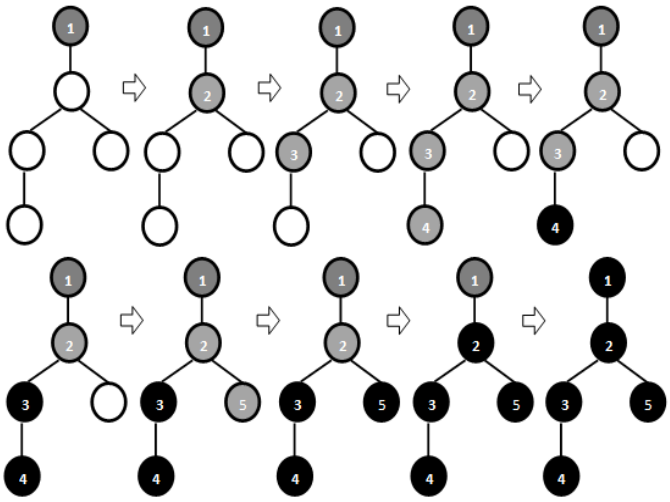

Figure 5.9- Recherche des CFCs en utilisant l'algorithme de Tarjan

Algorithme 5.2- Algorithme de Tarjan

```
Initialisation
   P=pile_vide ; n=0 ; k=0
   Pour chaque sommet s faire
      Découverte[s]=0 ; Index[s]=0 ; dansPile[s]=Faux ; comp[s]=0
   Fin Pour
Calcul de toutes les CFCs
   Pour chaque sommet s faire
      Si Découverte [s]=0
         CFC(s)
      Fin Si
   Fin Pour
Calcul de la CFC d'un sommet
   Procédure CFC(x:sommet)
      empiler(x,P) ; dansPile[x]=Vrai ; n=n+1; Découverte [x]=n ; m= Découverte [x]
      Pour chaque successeur y de x faire
         Si Découverte [y]=0
            CFC(y)
            m=min(Découverte [x],Index[y])
         Sinon
            Si dansPile[y]
               m=min(m, Découverte [y])
            Fin Si
         Fin Si
      Fin Pour
      Index[x]=m
      Si Index[x]= Découverte [x]
         k=k+1
         Répéter
            y=sommetPile[P] ; depiler(P) ; dansPile[y]=Faux comp[y]=k
```

```
        Jusqu'à y=x
    Fin Si
Fin Procédure
```

Pendant l'exécution de l'algorithme les composantes sont rangées dans l'ordre de leur découverte, et chaque état est empilé exactement une seule fois, ce qui donne un temps d'exécution linéaire.

L'algorithme Kosaraju (1978 non publié) représente une autre méthode de calcul des CFCs, le professeur d'algorithmique Kosaraju qui a publié ses notes de cours sur l'algorithme de Tarjan, il a pu improviser son propre algorithme qui se base aussi sur le parcours en profondeur mais avec un double passe contrairement à l'algorithme de Tarjan.

5.2.2 Classification en niveaux

Pour classifier les CFCs résultantes en niveaux, on considère les CFCs comme des sommets dans les autres techniques d'hiérarchisation citées auparavant. Ainsi, on associe à G un graphe réduit [87] $G_R(X_R, U_R)$ dont les sommets X_R correspondent aux CFCs de G : $X_R = \bigcup_{i=1,2,\ldots} C_i$ et U_R correspond aux arcs de G dont les extrémités n'appartiennent pas à la même CFC, ceci implique qu'on a un arc reliant deux sommets dans X_R , s'il existe un sommet $x_i \in C_i$ et $x_j \in C_j$ avec $p(x_j \mid x_i) > 0$ et $i \neq j$. Le graphe réduit correspondant au graphe de la figure 5.7 peut être illustré par :

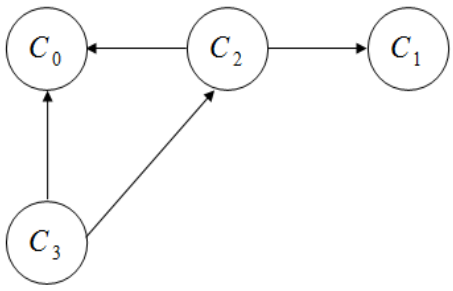

Figure 5.10- Graphe réduit

En appliquant la méthode d'hiérarchisation en sommets ou bien en niveaux, on aura les résultats présentés dans la figure suivante :

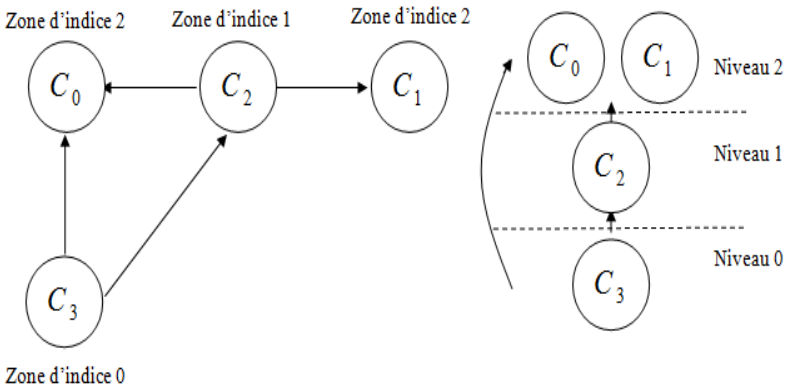

Figure 5.11- Hiérarchisation des CFCs

La technique de décomposition choisie prendra la même direction mais en inversant l'ordre :

- Suivant la méthode d'hiérarchisation des sommets :
  - Le niveau l=0 combine tous les sommets (Les CFCs) de degré extérieur nul.
  - Le niveau l={1, 2, … (L-1)} contienne tous les sommets (les CFCs) qui ont tous leurs successeurs dans les niveaux inferieur à l.
- Suivant la méthode d'hiérarchisation des niveaux :
  - L'équation (5.1) peut être définie par :

$$h(C_i) > h(C_j) \Leftrightarrow \text{S'il existe un chemin de } C_i \text{ vers } C_j \qquad (5.5)$$

  - L'équation (5.2) peut être définit par :

$$\forall z : 0 \le z \le (L-1) : \exists x : h(x) = z \qquad (5.6)$$

Donc on aura la décomposition suivante :

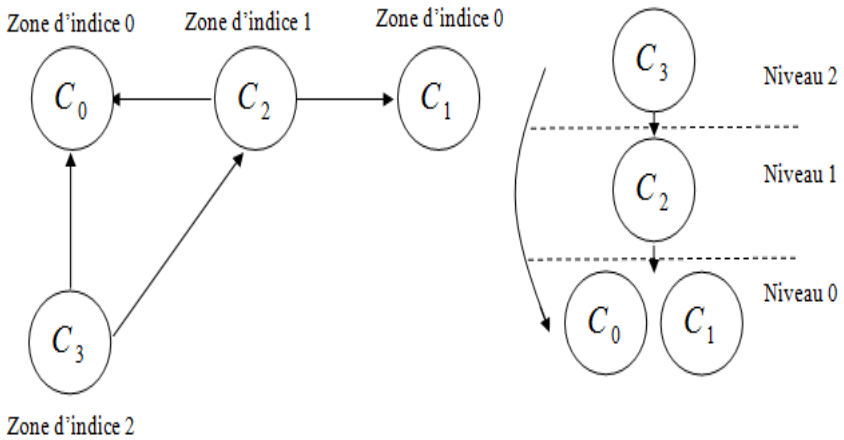

Figure 5.12- Hiérarchisation des CFCs dans l'ordre inverse

Ainsi, après avoir partitionné l'espace d'état du modèle en CFCs, $C_1,C_2,...,C_H$, on construit par récurrence les niveaux du graphe G(X, U). De telle sorte, le premier niveau L_0 ne contient que les CFCs fermées C_i (De degré extérieur nul). Le $(n-1)^{éme}$ niveau : L_{n-1} est formé par les CFCs C_i dont les arcs y sortant sont dans les niveaux inférieurs $L_{n-2},...,L_0$.

***Remarque 5.3* :** Nous pouvons reformuler les deux remarques 5.1 et 5.2 :

- Les deux classes C_i et C_j ont le même indexe de région ou bien appartiennent au même niveau si il n'existe pas une probabilité non nul d'un état de C_i à un état de C_j et vise versa;
- Si une classe C_i appartient à un niveau L_k , il existe un chemin d'un état de C_i à un état de C_j alors la classe C_j se trouve dans un niveau inférieur strictement à L_k.

Théorème 5.1 : Soit C_i une classe dans le niveau L_{n-1} alors C_i est fermée pour le MM restreint à l'espace d'état $S \mid L_{n-2},...,L_0$ [2].

Preuve : elle est immédiate par définition de niveau.

Reprenons l'exemple du graphe G cité dans la figure 5.1, alors :

L_2	C_3
L_1	C_2
L_0	C_0 et C_1

Tableau 5.1- Décomposition en niveaux

On note par C_{pk} la $k^{éme}$ CFC du niveau p, avec K(p) représente le nombre maximal des CFCs dans le niveau p. Prenons l'exemple du tableau (5.1) , en utilisant la nouvelle notation par exemple pour le niveau L_0 , la classe C_{01} et C_{02} correspond respectivement aux classes C_0 et C_1 , dans ce cas la variable k prend deux valeurs {1, 2} et le nombre maximal des CFCs du niveau L_0 , K(0)= 2 (pour les autres niveaux voir le tableau (5.2)).

L_2	C_{12}
L_1	C_{11}
L_0	C_{01} et C_{02}

Tableau 5.2- Décomposition en niveaux avec la nouvelle notation

5.2.3 Construction des PDMs restreints

On considère des PDMs à temps discret avec un espace d'états S et d'actions A finis. Après avoir décomposé l'espace d'états en CFCs et les avoir classées en n niveaux, l'étape suivante consiste à construire les modèles restreints à chaque CFC. On note par PDM_{pk}, le PDM restreint à la classe C_{pk}, $p = \{0,...,(n-1)\}$ et $k \in \{1,2,\ldots K(p)\}$. On construit ici par récurrence les PDMs restreints correspondants à chaque niveau L_p, p = 0, 1,…,(n-1).

- Les PDMs restreints au niveau L_0

Le modèle est définit par le quadruplet $(S_{0k}, A_{0k}, P_{0k}, R_{0k})$, pour tout $k \in \{1,2,\ldots,K(0)\}$, on note PDM_{0k}, le PDM restreint au classe C_{0k}. Le PDM_{0k} garde la définition d'espace d'état telle que $S_{0k} = C_{0k}$, il est bien défini et peut être résolu par un algorithme classique simple.

- Les PDMs restreints au niveau L_1

Pour tout $s \in C_{0h}$, $h \in \{1,2,\ldots,K(0)\}$ posons $V_{0h}(s)$ la valeur optimale calculée dans le niveau L_0. Pour tout $k \in \{1,2,\ldots,K(1)\}$, on note PDM_{1k}, le PDM restreint défini par le quadruplet $(S_{1k}, A_{1k}, P_{1k}, R_{1k})$:

- Espace d'état :

$$S_{1k} = C_{1k} \cup \{s' \in L_0 : p(s'|s,a) > 0 \text{ pour tout } s \in C_{1k}, a \in A(s)\} \tag{5.7}$$

- Espaces d'actions, pour $s, s' \in S_{1k}$:

$$A_{1k} = \begin{cases} A(s) \text{ si } s \in C_{1k} \\ \theta \text{ sinon} \end{cases} \tag{5.8}$$

- Probabilités de transition, pour tout $s \in S_{1k}$

$$p_{1k}(s'|s,a) = \begin{cases} p(s'|s,a) \text{ si } s \in C_{1k}, a \in A(s) \\ 1 \text{ si } s = s', s \notin C_{1k} \end{cases} \tag{5.9}$$

- Gains, pour tout $s \in S_{1k}$:

$$r_{1k}(s,a) = \begin{cases} r(s,a) \text{ si } s \in C_{lk} \\ (1-\gamma)V_{0h}(s) \text{ si } s \notin C_{1k}, \textit{pour } h \in \{1,...,K(0)\} \text{ et } s \in C_{0h} \end{cases} \tag{5.10}$$

- Les PDMs restreints au niveau L_p

Soit $S_p = \{\cup C_{mh}, m = 0,...,p-1$ et h=1,...,K(m)$\}$. Pour $s \in S_p$, posons $V_{mh}(s)$ la valeur optimale calculée dans les niveaux précédents. Pour h$\in\{1,2,\ldots,K(p)\}$, on note PDM_{pk}, le PDM restreint défini par le quadruplet $(S_{pk}, A_{pk}, P_{pk}, R_{pk})$:

- Espace d'état :

$$S_{pk} = C_{pk} \cup \Gamma(C_{pk}) \tag{5.11}$$

avec $\Gamma(C_{pk})$ est l'ensemble des successeurs nommé aussi états périphériques ou bien accessoires des états de C_{pk}.

- Espaces d'actions, pour tout $s \in S_{pk}$:

$$A_{pk}(s) = \begin{cases} A(s) & \text{si } s \in C_{pk} \\ \theta & \text{sinon} \end{cases} \tag{5.12}$$

- Probabilités de transition, pour $s, s' \in S_{pk}$:

$$p_{pk}(s'|s,a) = \begin{cases} p(s'|s,a) \text{ si } s \in C_{pk}, a \in A(s) \\ 1 \text{ si } s = s', s \notin C_{pk} \end{cases} \tag{5.13}$$

- Gains, pour tout $s \in S_{pk}$:

$$r_{pk}(s,a) = \begin{cases} r(s,a) \text{ si } s \in C_{pk} \\ (1-\gamma)V_{mh}(s) \text{ si } s \notin C_{pk}, \text{pour } s \in C_{mh} \text{ tels que h} \in \{1,..,K(m) \text{ et } m \in \{0,...,p-1\}\} \end{cases}$$

Remarque 5.4 : Pour chaque état accessoire s du niveau L_1, il existe h$\in\{1,2,\ldots,K(0)\}$ tel que $s \in C_{0h}$. Alors pour conserver sa valeur optimale $V_{0h}(s)$ qui est déjà calculée au niveau L_0, on doit lui assigner la récompense $r_{1k}(s,a) = (1-\gamma)V_{0h}(s)$.

On présente ici le résultat fondamental de cette section.

Théorème 5.2 : Si $V_{mh}(s)$ désigne la valeur optimale de l'état $s \in C_{mh}$ pour le PDM_{mh} alors $V_{mh}(s) = V(s)$ [2].

Preuve : La démonstration se fait par récurrence. Pour m=0, k$\in\{1,2,\ldots,K(0)\}$, les classes C_{0k} sont fermées. Pour tout $s \in C_{0k}$ la valeur optimale V est la solution unique de :

$$V(s) = \max_{a \in A(s)} \left\{ r(s,a) + \gamma \sum_{s' \in C_{0k}} p(s'|s,a) V(s') \right\} \tag{5.14}$$

La valeur optimale V_{0k} est la solution unique de :

$$V_{0k}(s) = \max_{a \in A_{0k}(s)} \left\{ r_{0k}(s,a) + \gamma \sum_{s' \in C_{0k}} p_{0k}(s'|s,a) V_{0k}(s') \right\} \tag{5.15}$$

En utilisant (5.14), (5.15) et le fait que $A_{0k}(\mathrm{s}) = A(\mathrm{s})$, $r_{0k}(\mathrm{s},\mathrm{a}) = r(\mathrm{s},\mathrm{a})$ et $P_{0k}(\mathrm{s}'|\mathrm{s},\mathrm{a}) = P(\mathrm{s}'|\mathrm{s},\mathrm{a})$ pour tout $\mathrm{s} \in C_{0k}$, nous avons $V_{0h}(s) = \mathrm{V}(\mathrm{s})$.

Soit $m > 0$, supposons que le résultat est vrai pour tout niveau précédant m, maintenant on montre que le résultat est vrai pour m. Soit $\mathrm{V}_{\mathrm{mk}}(\mathrm{s})$, $s \in C_{mk}$ la valeur optimale pour le PDM restreint PDM_{mk}, nous avons :

$$V_{mk}(s) = \max_{a \in A_{mk}(s)} \left\{ \begin{array}{l} r_{mk}(s,a) + \gamma \sum_{s' \in C_{mk}} p_{mk}(s'|s,a) V_{mk}(s') + \\ \gamma \sum_{s' \in (\mathrm{S}_{mk} | C_{mk})} p_{mk}(s'|s,a) V_{mk}(s') \end{array} \right\} \tag{5.16}$$

D'après l'hypothèse de récurrence $\mathrm{V}_{\mathrm{mk}}(\mathrm{s}')=\mathrm{V}(\mathrm{s}')$, sachant que la valeur V(s') est déjà calculée dans les niveaux précédents pour tout $s' \in (\mathrm{S}_{mk} \mid \mathrm{C}_{mk})$. Alors pour tout $s \in C_{mk}$, on a :

$$V_{mk}(s) = \max_{a \in A(s)} \left\{ \begin{array}{l} r(s,a) + \gamma \sum_{s' \in C_{mk}} p(s'|s,a) V_{mk}(s') + \\ \gamma \sum_{s' \in (\mathrm{S}_{mk} | C_{mk})} p(s'|s,a) V(s') \end{array} \right\} \tag{5.17}$$

Puisque $\mathrm{V}(\mathrm{s})$, $s \in S_{mk}$ est la solution unique de (5.17), donc :

$$\mathrm{V}_{\mathrm{mk}}(\mathrm{s})=\mathrm{V}(\mathrm{s}), \forall \mathrm{s} \in \mathrm{C}_{nmk} \tag{5.18}$$

Corollaire 5.1 : Soit π_{mk} une stratégie optimale du PDM_{mk} alors pour tout $s \in C_{mk}$, $\pi_{mk}(s)$ est une action optimale pour le PDM originel.

Preuve : Pour $s \in C_{mk}$, (D'après le théorème 5.2) nous avons :

$$\pi_{mk}(s) = \arg\max_{a \in A_{mk}(s)} \left\{ r_{mk}(s,a) + \gamma \sum_{s' \in S_{mk}} p_{mk}(s'|s,a) V_{mk}(s') \right\} \tag{5.19}$$

$$= \arg\max_{a \in A(s)} \left\{ r(s,a) + \gamma \sum_{s' \in S_{mk}} p(s'|s,a) V(s') \right\} \tag{5.20}$$

D'après la remarque 5.3, il y'a une dépendance entre les états d'un niveau avec les états des niveaux inférieurs. Cette dépendance permet d'utiliser seulement les états successeurs dans la mise à jour de la fonction valeur ainsi que dans la recherche de la stratégie optimale au lieu d'utiliser tous les états du système (Figure 5.13 et 5.14).

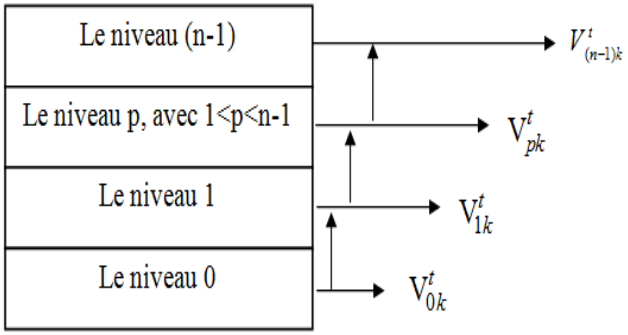

Figure 5.13- Dépendance minimale entre les niveaux pour le calcul de la fonction valeur

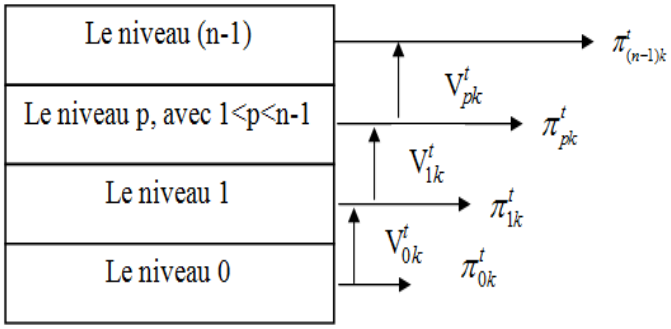

Figure 5.14- Dépendance minimale entre les niveaux pour le calcul des stratégies

Après avoir décomposé le système, on utilise l'un des algorithmes classiques tel que l'algorithme d'IV pour résoudre les PDMs restreints de façon hiérarchique. On commence par la résolution des PDMs restreints des niveaux inférieurs L_0 puis L_1 jusqu'à le niveau L_{n-1}. L'algorithme 5.3 expose l'algorithme d'IV Hiérarchique.

Algorithme 5.3- IV Hiérarchique

Pour chaque $p = 0 \rightarrow (n-1)$

Pour chaque $k = 0 \rightarrow K(p)$

$t \leftarrow 0$

Pour chaque $s \in C_{pk}$

$V^t_{pk}(s) = 0$

Fin pour

Répéter

$t \leftarrow t+1$

Pour chaque $s \in C_{pk}$ faire

$$V_{pk}^{t}(s) = \max_{a}\left\{ r_{pk}(s,a) + \gamma \sum_{s' \in S_{pk}} p_{pk}(s' \mid s,a) V_{pk}^{t-1}(s') \right\}$$

$$\pi_{pk}^{t}(s) = \arg\max_{a}\left\{ r_{pk}(s,a) + \gamma \sum_{s' \in S_{pk}} p_{pk}(s' \mid s,a) V_{pk}^{t-1}(s') \right\}$$

Fin pour

Jusqu'à $\max_{s} | V_{pk}^{t}(s) - V_{pk}^{t-1}(s) | < \varepsilon$

Retourner π_{pk}^{t}

Fin pour

Fin pour

5.3 Conclusion

Dans ce chapitre, nous avons montré les différentes étapes de **la technique de décomposition** inventée pour les **Problème Décisionnels de Markov** (PDMs) et la théorie des graphes leur principale source d'inspiration. Après avoir décomposé le système en **Composantes Fortement Connexes** (CFCs) et les partager en niveaux, nous avons procédé à la phase de construction des **PDMs restreints**. Ce chapitre sera la base du chapitre suivant, dont nous allons s'inspirer pour appliquer la technique de décomposition sur les **Modèles de Markov Cachés** (MMCs).

6 Décomposition des Modèles de Markov Cachés

La **technique de décomposition** détaillée dans le chapitre précédent est conçue particulièrement aux **Problèmes Décisionnels de Markov** (PDMs) sous différents critères. Dans ce chapitre nous allons valider la méthode de décomposition adoptée en calculant la complexité des algorithmes hiérarchiques. Nous allons apporter des modifications à cette technique pour pouvoir l'appliquer même aux **Modèles de Markov Cachés** (MMCs). Nous allons présenter les algorithmes **hiérarchiques** résultants : **Forward Hiérarchique**, **Backward Hiérarchique** et **Baum welch Hiérarchique** ainsi que les théorèmes et l'ordre de complexité qui valident l'efficacité de ces algorithmes.

6.1 Les Modèles de Markov Cachés restreints

Les PDMs sont les champs d'application de la méthode de décomposition proposée par Abbad, Boustique et Daoui [1, 2, 27, 28]. Dans ce chapitre, nous considérons le même principe en modifiant la méthode de décomposition pour l'avoir appliquer aux MMCs. Cette modification réside dans la façon de construction des MMCs restreints et les démarches générales de la résolution. Ces changements seront dirigés par le type d'algorithme de résolution utilisé.

6.2 Algorithmes de résolution hiérarchiques

6.2.1 Algorithme Forward Hiérarchique

L'algorithme IV et l'algorithme Forward cherchent à calculer respectivement la fonction valeur $V^t(i)$ et la variable Forward $\alpha_t(i)$ pour tout état i du modèle de façon itérative. La formule de mise à jour de la variable $\alpha_{t+1}(i)$ utilise des variables $\alpha_t(\mathrm{j})$ des états prédécesseurs j de l'état i contrairement à l'algorithme IV qui utilise les états successeurs de i pour calculer la fonction valeur $V^t(i)$. Dans un graphe orienté, on définit l'ensemble des prédécesseurs du sommet i par $\Gamma^{-1}(i)$ et l'ensemble des prédécesseurs pour tout état $i \in C_{pk}$ par $\Gamma^{-1}(C_{pk})$. Dans l'exemple suivant $\Gamma^{-1}(i) = \{j_2, j_4\}$.

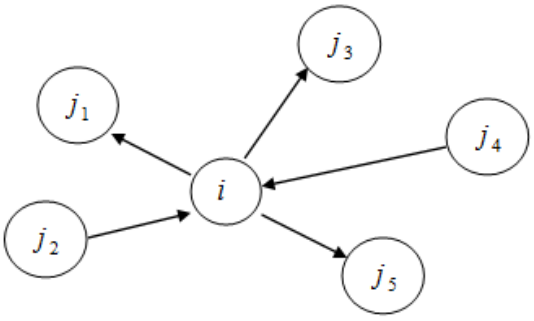

Figure 6.1- Prédécesseurs d'un état i donné

Cette différence entre ces deux algorithmes nous a donné l'idée principale qui sous-tend l'hiérarchisation de l'algorithme Forward. Nous construisons par récurrence les MMCs restreints correspondants à chaque niveau L_p , p = {n-1,…, 0}, nous débutons par le niveau supérieur L_{n-1} contrairement au cas des PDMs.

- Les MMCs restreints au niveau L_{n-1} : Pour $k \in \{1,2,...,K(n-1)\}$, le $MMC_{(n-1)k}$ désigne le modèle restreint à la classe $C_{(n-1)k}$. Il est définit par le quintuple $\{S_{(n-1)k}, \mathrm{O}_{(n-1)k}, \mathrm{A}_{(n-1)k}, \mathrm{B}_{(n-1)k}, \Pi_{(n-1)k}\}$ avec $S_{(n-1)k} = C_{(n-1)k}$ et $\mathrm{O}_{(n-1)k} = O = \{O_1, O_2, ..., O_M\}$. La probabilité de transition, la probabilité d'observation et les probabilités d'initialisation liées à l'espace d'état restreint $S_{(n-1)k}$ et l'espace d'observation $O_{(n-1)k}$ gardent la même définition que celles du MMC global, ainsi $\forall\, i, j \in S_{(n-1)k}$:

$$\mathrm{A}_{(n-1)k} = \left[a_{ij}\right]_{(n-1)\mathrm{k}} = A = \left[a_{ij}\right] \tag{6.1}$$

$$B_{(n-1)k} = \left[b_j(m)\right]_{(n-1)\mathrm{k}} = B = \left[b_j(m)\right] \tag{6.2}$$

$$\Pi_{(n-1)k} = \left[\pi_i\right]_{(n-1)\mathrm{k}} = \Pi = \left[\pi_i\right] \tag{6.3}$$

  Comme le modèle restreint $MMC_{(\mathrm{n}-1)k}$ est bien défini, il peut être résolu par l'algorithme Forward classique.
- Les MMCs restreints au niveau $L_p, p < (n-1)$: Pour $\mathrm{k} \in \{1,2,\ldots,\mathrm{K(p)}\}$, le MMC_{pk} désigne le MMC restreint défini par le quintuple $\{S_{pk}, O_{pk}, A_{pk}, B_{pk}, \prod_{pk}\}$:
  - L'espace d'états $S_{pk} = C_{pk} \cup \Gamma^{-1}(C_{pk})$ représente l'union des états de la classe C_{pk} et l'ensemble de ces prédécesseurs $\Gamma^{-1}(C_{pk})$;
  - La matrice des probabilités de transition $A_{pk} = \left[a_{ij}\right]_{pk} = A = \left[a_{ij}\right]$ pour $\mathrm{j} \in C_{pk}$, $\mathrm{i} \in \mathrm{S}_{pk}$; avec $a_{ij} = p(S_{t+1} = j \mid S_t = i)$;
  - Les observations selon M symboles $O_{pk} = \{O_1, O_2, ..., O_M\} = O$;

- La matrice des probabilités initiales $\Pi_{pk} = [\pi_i]_{pk} = \Pi = [\pi_i]$; pour $i \in C_{pk}$, avec $\pi_i = p(S_t = i)$;
- La matrice des probabilités d'observations $B_{pk} = \left[b_j(m)\right]_{pk} = B = \left[b_j(m)\right]$ pour $j \in C_{pk}$ avec $b_j(m) = p(O_t = o_m \mid S_t = j)$.

Soit $i \in C_{pk}$, la variable $\alpha_{t,pk}(i)$ présente la variable Forward à l'instant t correspondante à l'état i qui appartient à la classe C_{pk}, elle représente la probabilité d'avoir généré la séquence $O_{pk} = \{O_1, O_2, ..., O_t\}$ et d'atteindre l'état i à l'instant t.

Nous présentons notre résultat fondamental de cette section :

Lemme 6.1 : Soit $j \in C_{pk}$, la variable Forward pour l'état j à l'instant t+1 est donnée par :

$$\alpha_{t+1,pk}(j) = \left[\sum_{i \in S_{pk}} \alpha_{t,pk}(i) \times a_{ij} \right] b_j(o_{t+1}) \tag{6.4}$$

Preuve : L'équation 2.2 permet de calculer la variable Forward $\alpha_{t+1}(j)$ pour $t \in [1, T-1]$ et $j \in [1, N]$ de manière inductive : $\alpha_{t+1}(j) = \left[\sum_{i \in S} \alpha_t(j)\, a_{ij} \right] b_j(o_{t+1})$.

Pour tout $j \in S$, d'après la définition de l'ensemble S_{pk}, nous avons $a_{ij} = 0$ pour tout $i \in (S \mid S_{pk})$, donc : $\left[\sum_{i \in (S \mid S_{pk})} \alpha_t(j)\, a_{ij} \right] b_j(o_{t+1}) = 0$.

Ainsi, nous pouvons reformuler l'équation 2.2 comme suit:

$$\alpha_{t+1}(j) = \left[\sum_{i \in S_{pk}} \alpha_t(j)\, a_{ij} \right] b_j(o_{t+1}) \tag{6.5}$$

D'après la définition du modèle restreint MMC_{pk}, pour tout p=0,…, (n-1) et $k = \{0, ..., K(p)\}$, nous avons : $A_{pk} = A$, $O_{pk} = O$, $\Pi_{pk} = \Pi$ et $B_{pk} = B$, ainsi :

$$\alpha_{t+1,pk}(j) = \left[\sum_{i \in S_{pk}} \alpha_{t,pk}(i) \times a_{ij} \right] b_j(o_{t+1}), \; j \in C_{pk} \tag{6.6}$$

L'algorithme Forward Hiérarchique (Algorithme 6.1) est l'algorithme résultant de l'application de la méthode de décomposition modifiée sur l'algorithme Forward. Il consiste principalement à faire une initialisation et une mise à jour de la variable $\alpha_{t,pk}(i)$ pour chaque niveau p en commençant par le niveau supérieur L_{n-1} jusqu'à l'arrivée au niveau L_0.

Algorithme 6.1- Forward Hiérarchique

Etape1: Initialisation

Pour $p = 0,...,(n-1)$ et $k = \{0,...,\mathrm{K(p)}\}$; soit $i \in C_{pk}$

$$\alpha_{1,pk}(i) = \pi_i \times b_i(o_1)$$

Etape 2: Induction

Pour tout $t \in [t, T-1]$, p=(n-1), …, 0 et k=1, 2, …, K(p) ; soit $j \in C_{pk}$

$$\alpha_{t+1,pk}(j) = \left[\sum_{i \in S_{pk}} \alpha_{t,pk}(i) \times a_{ij} \right] b_j(o_{t+1})$$

Etape 3 : Terminaison

$$\mathrm{p(O \mid \lambda)} = \sum_{p=0}^{n-1} \sum_{k=1}^{K(p)} \sum_{i \in C_{pk}} \alpha_{T,pk}(i)$$

L'utilisation des prédécesseurs dans le calcul de la variable Forward engendre une dépendance entre les niveaux (Figure 6.2). En effet, le calcul de $\alpha_{t+1,pk}(j)$ ne nécessite que les valeurs $\alpha_{t,pk}(i)$ avec $i \in \Gamma^{-1}(i)$ et qui sont nécessairement calculées dans les niveaux supérieurs à p.

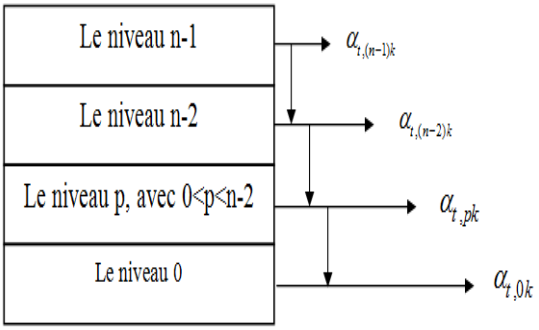

Figure 6.2- Dépendance minimale entre les niveaux pour le calcul de la variable Forward

6.2.2 Algorithme Backward Hiérarchique

Contrairement à l'algorithme Forward Hiérarchique l'algorithme Backward Hiérarchique construit des MMCs restreints en commençant par le niveau L_0 et nous passons vers le niveau L_{n-1} (Figure 6.3). Ce qui est similaire au principe de construction de l'algorithme d'IV.

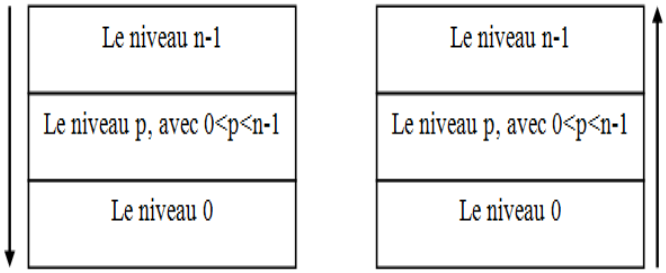

Figure 6.3- Construction des modèles restreints pour l'algorithme Forward Hiérarchique Backward Hiérarchique

Contrairement à l'algorithme Forward, la formule de la mise à jour de la variable $\beta_t(i)$ utilise les variables $\beta_{t+1}(i)$ des états successeurs de l'état traité i au lieu des états prédécesseurs. Nous construisons par récurrence les MMCs restreints correspondants à chaque niveau L_p, p ={0, 1,…, n-1}. En commençant par le niveau L_0.

- Les MMCs restreints au niveau L_0 : Pour k ={1,…, K(0)}, le MMC_{0k} présente le modèle restreint à la classe C_{0k}. Il est défini de la même façon que le modèle global :

$$S_{0k} = C_{0k} \tag{6.7}$$

$$\mathrm{A}_{0k} = \left[a_{ij}\right]_{0k} = A = \left[a_{ij}\right] \tag{6.8}$$

$$B_{0k} = \left[b_j(m)\right]_{0k} = B = \left[b_j(m)\right] \tag{6.9}$$

$$\Pi_{0k} = \left[\pi_i\right]_{0k} = \Pi = \left[\pi_i\right] \tag{6.10}$$

  Du même le MMC_{0k}, $\forall\, k \in \{1,2,\ldots K(0)\}$ est bien défini et peut être résolu par l'algorithme Backward classique.

- Les MMCs restreints au niveau L_p, p > 0 : Pour k$\in$ {1,2,…,K(p)}, le MMC_{pk} présente le MMC restreint à la classe C_{pk} défini par le quintuple $\{S_{pk}, O_{pk}, A_{pk}, B_{pk}, \prod_{pk}\}$:
  - L'espace d'états $S_{pk} = C_{pk} \cup \Gamma(C_{pk})$ représente l'union des états de la classe C_{pk} et l'ensemble de ces prédécesseurs $\Gamma(C_{pk})$;
  - La matrice des probabilités de transition $A_{pk} = \left[a_{ij}\right]_{pk} = A = \left[a_{ij}\right]$ pour $j \in S_{pk}$, $i \in C_{pk}$; avec $a_{ij} = p(S_{t+1} = j \mid S_t = i)$;
  - Les observations selon M symboles $O_{pk} = \{O_1, O_2, ..., O_M\} = O$;
  - La matrice des probabilités initiales $\Pi_{pk} = \left[\pi_i\right]_{pk} = \Pi = \left[\pi_i\right]$; pour $i \in C_{pk}$, avec $\pi_i = p(S_t = i)$;

- La matrice des probabilités d'observations $B_{pk} = \left[b_j(m) \right]_{pk} = B = \left[b_j(m) \right]$ pour $j \in S_{pk}$ avec $b_j(m) = p(O_t = o_m \mid S_t = j)$.

Nous notons par $\beta_{t,pk}(i)$, la variable Backward à l'instant t correspondant à la classe C_{pk} qui définit la probabilité d'avoir généré la séquence $O_{pk} = \{O_{t+1}, ..., O_T\}$ en partant à l'instant t de l'état $i \in C_{pk}$.

Lemme 6.2: Soit l'état $i \in C_{pk}$, la variable Backward à l'instant t pour l'état i est donnée par :

$$\beta_{t,pk}(i) = \left[\sum_{j \in S_{pk}} a_{ij} b_j(o_{t+1}) \beta_{t+1,pk}(j) \right] \tag{6.11}$$

Preuve : la démonstration reste le même que celle du lemme 6.1, en se basant sur l'équation classique (Equation 2.4) pour le calcul de la variable Backward $\beta_t(i)$ pour $t \in [T-1, 1]$ et $i \in [1, N]$.

L'algorithme Backward Hiérarchique (Algorithme 6.2) proposé calcule la variable Backward de façon hiérarchique, en commençant par le niveau inférieur L_0 et aborde successivement les niveaux supérieurs jusqu'à l'arrivée au niveau L_n.

Algorithme 6.2- Backward Hiérarchique

Etape 1: Initialisation

Pour $p = 0, ..., (n-1)$ et $k = \{0, ..., K(p)\}$; soit $i \in C_{pk}$

$$\beta_{T,pk}(i) = 1$$

Etape 2: Induction

Pour tout $t \in [T-1, 1]$, $p = 0, ..., (n-1)$ et $k = \{0, ..., K(p)\}$; soit $i \in C_{pk}$

$$\beta_{t,pk}(i) = \left[\sum_{j \in S_{pk}} a_{ij} b_j(o_{t+1}) \beta_{t+1,pk}(j) \right]$$

Soit $j \in C_{pk}$, le calcul de la variable $\beta_{t,pk}(j)$ nécessite les valeurs des variables $\beta_{t+1,pk}(i)$ avec $i \in \Gamma(C_{pk})$ qui sont forcément calculées dans les niveaux inférieurs.

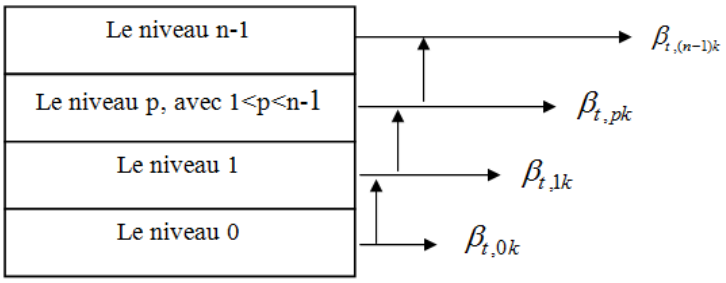

Figure 6.4- Dépendance minimale entre les niveaux pour le calcul de la variable Backward

6.2.3 Algorithme Baum-welch Hiérarchique

Comme il est indiqué dans le chapitre 2, l'algorithme Baum-Welch permet de réestimer des paramètres du modèle caché. Il se base le long de cette ré-estimation, sur les résultats obtenus de l'exécution de l'algorithme Forward et Backward. Ainsi, toute amélioration de ces deux derniers algorithmes représente une amélioration de l'algorithme Baum-Welch.

6.3 Analyse de la complexité

Dans ce paragraphe, nous validons les résultats obtenus en se basant sur une étude de la complexité des algorithmes hiérarchiques proposés dans ce chapitre et dans le chapitre précédant en utilisant la méthode de décomposition cité dans le chapitre précédant ou après la modification faite sur cette méthode dans ce chapitre . Ces algorithmes sont tous récursifs et ils rentrent dans la catégorie des algorithmes adoptant la méthode « diviser pour régner » comme principe de conception, elle permet de diviser l'entrée en plusieurs sous problèmes, les résoudre séparément, puis les tricoter pour résoudre le problème d'origine. Il existe en littérature deux théorèmes facilitant le calcul de la complexité des algorithmes hiérarchiques : « le Théorème Général » (Master Theorem) et le théorème « d'Akra-Bazzi ». Dans ce qui suit, nous allons présenter un bref résumé de ces théorèmes.

En supposant que le problème de taille n est divisé en a sous problèmes de taille $\frac{n}{b}$, nous résolvons ces sous-problèmes puis nous recombinons les solutions partielles en une solution du problème initial. La complexité en temps du système globale T(n) est naturellement la somme de la complexité en temps de calcul pour tous les a sous problèmes et le coût de gestion en dehors des appels récursifs, ce qui est résumé par le temps gaspillé dans l'étape de la décomposition D(n) et le temps de fusion F(n) des solutions de chaque sous problème. Ainsi, la complexité en temps de ces algorithmes est définit par une relation de récurrence de type :

$$T(n) = aT(\frac{n}{b}) + F(n) + D(n) \qquad (6.12)$$

Le temps de fusionnement est généralement négligeable devant le temps d'exécution des sous problèmes :

$$F(n) \simeq 0 \tag{6.13}$$

Le Théorème Général suivant ne s'applique que si les sous problèmes sont de même taille [32, 91]. Il permet d'exprimer la complexité sans passer par un développement explicite de (6.12) :

Théorème 6.1 (Théorème Général) : La complexité de la formule suivante :

$$T(n) = aT(\frac{n}{b}) + D(n) \text{ avec } a \geq 1 \text{ et } b > 1 \tag{6.14}$$

est catégorisée en trois cas :

- Si $D(n) = O(n^c)$ avec $c < \log_b(a)$, alors :

$$T(n) = O(n^{\log_b(a)}) \tag{6.15}$$

- Si $\mathrm{D(n)} = \mathrm{O}(n^c \log^k(\mathrm{n}))$ pour une constante $k \geq 0$ et $c = \log_b(a)$, alors :

$$T(n) = O(n^c \log^{k+1} n) \tag{6.16}$$

- Si $D(n) = O(n^c)$ avec $c > \log_b(a)$ et s'il existe une constante $k < 1$ telle que $\mathrm{aD}(\frac{n}{b}) \leq \mathrm{kD(n)}$, alors :

$$T(n) = O(D(n)) \tag{6.17}$$

Reprenons l'exemple de la recherche par dichotomie, cité au chapitre 1, et qui rentre dans la catégorie des algorithmes « diviser pour régner ». La recherche par dichotomie suit la logique de diviser le tableau de taille n à chaque instant en deux, jusqu'à atteindre la valeur recherchée. La décomposition du tableau en deux est une opération élémentaire d'ordre 1, ainsi la formule de la complexité à résoudre s'écrit dans ce cas sous la forme suivante :

$$\mathrm{T(n)} = \mathrm{T}(\frac{n}{2}) + D(n) = \mathrm{T}(\frac{n}{2}) + O(1) \tag{6.18}$$

Nous avons d'une part a=1 et b=2 et d'une autre part $D(n) = 1$, alors forcément c=0, ce qui montre l'égalité des deux termes $\log_2(1) = \mathrm{c} = 0$. Clairement, nous somme dans le deuxième cas du théorème 6.1. Il reste alors à prouver qu'il existe une constante $k \geq 0$ qui réalise cette égalité $\mathrm{D(n)} = \mathrm{O}(n^c \log^k(\mathrm{n})) = 1$, ce qui n'est pas vrai que si k=0 :

$$D(n) = O(n^c \log^k(n)) = O(n^0 \log^0(n)) = 1 \tag{6.19}$$

Nous obtenons alors:

$$T(n) = O(n^c \log^{k+1} n) = O(n^0 \log^{0+1} n) = \log(n) \tag{6.20}$$

D'après la définition de la méthode de décomposition que nous avons adoptée, le partitionnement des CFCs ne se fait pas de façon systématique de telle sorte que nous gardons une taille commune pour toutes les CFCs, cette caractéristique montre l'inapplicabilité de ce théorème dans notre cas. Le théorème d'Akra Bazzi (Appelé aussi méthode d'Akra-Bazzi) est publié en 1998 [3] par M. Akra et L. Bazziqui qui est une généralisation du Théorème Général au niveau de la condition d'égalité des sous ensembles résultants après la décomposition ce qui s'adapte à notre besoin.

Nous décomposons le modèle en CFCs, $C_{i=1,\ldots,H}$, la taille de la $i^{éme}$ classe est $\frac{n}{b_i}$, alors nous pouvons reformuler autrement le temps d'exécution consommé par les sous problèmes ou bien par les CFCs:

$$aT(n) = \sum_{i=1}^{H} T_i(\frac{n}{b_i}) \tag{6.21}$$

Avec $T_i(\frac{n}{b_i})$ exprime le temps d'exécution de la $i^{éme}$ classe, et en rappelant que H est le nombre des CFCs. Notamment après la décomposition nous pouvons avoir un nombre a_i de CFCs ayant la même taille $c_i n$ ce qui divise les classes en k catégories. L'équation (6.22) exprime une autre forme de l'équation (6.25) , avec $c_i = \frac{1}{b_i} \in]0,1[$

$$aT(n) = \sum_{i=1}^{k} a_i T(c_i n) \tag{6.22}$$

Théorème 6.2 (Akra Bazzi) : Soit l'équation suivante (6.23), avec p la racine réelle et unique de l'équation $\sum_{i=1}^{k} a_i b_i^p = 1$:

$$T(n) = \sum_{i=1}^{k} a_i T(c_i n) + D(n)\ ,\ n > 1 \tag{6.23}$$

La solution de cette équation peut être écrite sous la forme suivante :

$$T(n) = O(n^p(1 + \int_1^n \frac{D(u)}{u^{p+1}} d(u)) \tag{6.24}$$

Comme il est montré dans le théorème 6.2, nous devons chercher la valeur de la variable p pour calculer l'intégrale dans l'équation (6.24). Dans les trois algorithmes hiérarchiques proposés, nous pouvons constater que le calcul soit de la fonction valeur, la variable Forward ou bien la variable Backward ne se fait que pour les états de la CFC et non pas sur la totalité

des états restreints, ce qui implique que la seule valeur possible de la variable p est 1. Alors l'équation (6.28) devient :

$$T(n) = O(n(1 + \int_1^n \frac{D(u)}{u^2} d(u)) \tag{6.25}$$

En utilisant l'algorithme de Tarjan, la fonction D(u) est de complexité linéaire, en conclusion, l'intégrale (6.29) devient :

$$T(n) = \Theta(n(1 + \int_1^n \frac{u}{u^2} du)) = \Theta(n + n\int_1^n \frac{1}{u} du) = n\log(n) \tag{6.26}$$

Ainsi la complexité résultante est d'ordre $n\log(n)$, qui est incontestablement plus petit que la complexité quadratique $O(n^2)$ dans la forme classique des trois algorithmes traités.

6.4 Conclusion

Dans ce chapitre, nous avons détaillé les différents **algorithmes hiérarchiques** résultants de l'application de la **méthode de décomposition** modifiée sur les algorithmes classiques des Modèles de Markov Cachés. Dans la phase de validation, les résultats obtenus au niveau de la **complexité** sont encourageants par rapport aux algorithmes classiques. Le chapitre suivant sera consacré à l'hybridation de la technique de décomposition avec **les algorithmes itératifs** d'accélération de convergence basés sur le principe de **Gauss-Seidel** pour la résolution des **Problèmes Décisionnels de Markov** (PDMs).

7 Techniques de décomposition et d'accélération de la convergence

Dans les deux chapitres précédents, nous avons introduit les fondements de la méthode de décomposition des deux **Modèles Markoviens** (MMs), les **Problèmes Décisionnels de Markov** (PDMs) et **les Modèles de Markov Cachés** (MMCs).

Les méthodes itératives appliquées à la résolution des PDMs telles que **Gauss-Seidel** et **Jacobi** visent à diminuer le temps de convergence en améliorant l'équation de récursivité, elles commencent par une valeur approchée de la solution désirée, puis augmentent la précision par l'application itérée d'une procédure en utilisant instantanément les résultats obtenus en cours, ce qui diminue le temps de convergence qui représente une point critique pour les autres méthodes itératives classiques.

Dans ce chapitre nous allons proposer une approche d'hybridation de la méthode de Gauss-Seidel et la méthode de décomposition adoptée après l'avoir modifiée. Ainsi, nous allons proposer l'algorithme **IV-PGS Hiérarchique** résultant et nous allons valider leur efficacité. Nous allons montrer l'efficacité de notre approche au niveau de la convergence en comparant le temps d'exécution de IV-PGS Hiérarchique avec **IV-PJ Hiérarchique.**

7.1 Méthodes itératives

Les algorithmes classiques de résolution des PDMs mentionnés au chapitre 3, l'algorithme d'IV et l'algorithme d'IP, rentrent dans la catégorie des méthodes itératives. Ce genre de méthodes est utilisable pour la résolution des problèmes de type système d'équations et les problèmes d'optimisation quand les méthodes directes sont inapplicables ou coûteuses. Bien que les méthodes itératives sont simples en programmation et n'ont pas besoin d'un grand espace mémoire, elles sont plus complexes et nécessitent un grand temps de convergence vers la solution souhaitée.

Parmi les objectifs capitales des méthodes itératives est la résolution des équations linéaires, soit S un système d'équations linéaires (7.1) de n équations à n inconnues $(x_1, x_2, ..., x_n)$:

$$S : \begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ ... \quad + ... \quad + ... + \quad ... \quad = ... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \end{cases} \tag{7.1}$$

Sa forme matricielle peut être résumée par:

$$Ax = b \tag{7.2}$$

Avec A une matrice d'ordre n à coefficients a_{ij}, dont tous les éléments diagonaux a_{ii} sont tous non-nuls. Le vecteur colonne x de composantes x_i est appelé solution du système. Le vecteur colonne B représente le second membre avec n composantes b_i.

Dans la littérature, il existe plusieurs méthodes classiques pour la résolution de ce type de systèmes [16, 67, 105]. Dans ce qui suit nous allons détailler les deux méthodes classiques Jacobi et Gauss Seidel qui se basent sur le principe de point fixe.

Les méthodes de point fixe [67] utilisent le fait que le problème f(x)=0 peut être exprimée en un problème équivalent x=g(x), trouver les zéros d'une fonction f revient à approcher les points fixes de la fonction g. La méthode la plus utilisée pour la précision du point fixe c'est le calcul itératif qui vise à construire pour $x^{(0)}$ donnée la suite de solutions approximatives $x^{(k+1)}$, qui se rapprochent de la solution exacte x, à partir de la relation de l'équation (7.3) nommée itération de point fixe.

$$x^{(k+1)} = g(x^{(k)}),\ k \geq 0 \tag{7.3}$$

Nous pouvons reformuler le système (7.2). On décompose la matrice A en deux matrices M et N :

$$A = M - N \tag{7.4}$$

Où M est une matrice inversible. Ainsi, en remplaçant la matrice A (7.4) dans l'équation (7.2), le système peut être s'écrit sous la forme :

$$Mx = Nx + b \tag{7.5}$$

On utilise le principe de la méthode de point fixe, nous obtenons :

$$x = M^{-1}Nx + M^{-1}b = g(x) \tag{7.6}$$

Alors :

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b \tag{7.7}$$

7.1.1 Méthode de Jacobi

La méthode de Jacobi est l'une des méthodes itératives conçues pour la résolution des systèmes linéaires. Considérons un système d'équations linéaires d'ordre n, sa forme matricielle est présentée respectivement dans l'équation (7.1) et (7.2). Soit :

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \tag{7.8}$$

La méthode de Jacobi suit la même logique de décomposition citée dans l'équation (7.4), notant que la matrice M = D est une matrice inversible et la matrice N = E + F :

$$A = D - E - F \tag{7.9}$$

Avec :

- La matrice D est la matrice diagonale de A :

$$D = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} \tag{7.10}$$

- La matrice E (F) est la matrice triangulaire inférieure (supérieure) de diagonale nulle ;

$$E + F = \begin{bmatrix} 0 & -a_{12} & \dots & -a_{1n} \\ -a_{21} & 0 & \dots & -a_{2n} \\ \dots & \dots & \dots & \dots \\ -a_{n1} & -a_{n2} & \dots & 0 \end{bmatrix} \tag{7.11}$$

En utilisant ces notations, on peut écrire respectivement le système dans l'équation (7.5) et (7.6) sous la forme :

$$Dx = (E + F)x + b \tag{7.12}$$

$$x = D^{-1}((E + F)x + b) \tag{7.13}$$

Le résultat du produit des deux matrices $D^{-1}(E + F)$ est alors appelée matrice de Jacobi. L'utilisation de D^{-1} qui représente l'inverse de la matrice diagonale de A; justifie la condition sur les éléments diagonaux a_{ii} de A qu'on a supposés tous non nuls.

Les deux équations suivantes expriment l'itération de Jacobi qui est formalisée avec l'algorithme 7.1:

$$x^{(k+1)} = D^{-1}((E + F)x^{(k)} + b) \tag{7.14}$$

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1 \text{ et } j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}, \ pour\ i = 1, 2, \dots, n \tag{7.15}$$

A chaque itération, l'algorithme de Jacobi nécessite le stockage de deux vecteurs $x^{(k+1)}$ et $x^{(k)}$. Explicitons l'élément diagonal de chaque ligne et réécrivons le système S de la façon suivante :

$$S : \begin{cases} x_1 = \dfrac{b_1 - (a_{12}x_2 + ... + a_{1n}x_n)}{a_{11}} \\ x_2 = \dfrac{b_2 - (a_{21}x_1 + ... + a_{2n}x_n)}{a_{22}} \\ ... \\ x_n = \dfrac{b_n - (a_{n1}x_1 + ... + a_{nn-1}x_{n-1})}{a_{nn}} \end{cases} \quad (7.16)$$

Remarque 7.1 : pour $i = 1,...,n$, l'indépendance du calcul des valeurs x_i dans chaque itération rend la méthode de Jacobi facilement parallélisable .

Algorithme 7.1- Jacobi

Choix d'une solution initiale $x^{(0)} \in R^n$

Pour chaque k=0,1… jusqu'à la convergence faire

Pour chaque i=1, …, n faire

$$x_i^{(k+1)} = \frac{(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)})}{a_{ii}}$$

Fin pour

Fin pour

On commence par une estimation de la solution initiale $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$. On remplace par la suite ces valeurs dans le côté droit des équations réécrites (7.16) pour obtenir la première approximation $x^{(1)} = (x_1^{(1)}, x_2^{(1)}, ..., x_n^{(1)})$, ce qui permet d'accomplir une itération. De la même façon, la deuxième approximation $x^{(2)} = (x_1^{(2)}, x_2^{(2)}, ..., x_n^{(2)})$ est calculée en remplaçant cette fois ci la première approximation résultante $x^{(1)}$ dans la même équation. Par itérations répétée, on forme une suite d'approximations $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)})$, pour $k = 1, 2, ...$

Exemple 7.1 : Prenons l'exemple d'un système à trois inconnus suivant:

$$\begin{aligned} 10x_1 + 2x_2 - x_3 &= 7 \\ x_1 + 8x_2 + 3x_3 &= -4 \\ -2x_1 - x_2 + 10x_3 &= 9 \end{aligned}$$

Nous pouvons réécrire le système comme suit :

$$x_1 = \frac{7}{10} - \frac{2}{10}x_2 + \frac{1}{10}x_3$$
$$x_2 = -\frac{4}{8} - \frac{1}{8}x_1 - \frac{3}{8}x_3$$
$$x_3 = \frac{9}{10} + \frac{2}{10}x_1 + \frac{1}{10}x_2$$

La première approximation pour des valeurs initiales $(x_1^{(0)}, x_2^{(0)}, x_2^{(0)}) = (0,0,0)$ donne :

$$x_1^{(1)} = 0.7,\ x_2^{(1)} = -0.5\ et\ x_3^{(1)} = 0.9$$

En continuant les itérations, nous obtenons les approximations citées au dessous:

N	K=0	K=1	K=2	K=…
$x_1^{(k)}$	0	0.7	0.89	
$x_2^{(k)}$	0	-0.5	-0.925	
$x_3^{(k)}$	0	0.9	0.99	

Tableau 7.1- Résultats par itération en utilisant la méthode de Jacobi

Dans la section suivante, nous allons éclaircir la méthode de Gauss-Seidel qui représente un raffinement de la méthode de Jacobi.

7.1.2 Méthode de Gauss-Seidel

La méthode de Gauss-Seidel est nommée par Carl Friedrich Gauss et Philipp Ludwig von Seidel, elle a été mentionnée dans une lettre privée de Gauss à son élève Gerling en 1823 et n'est publiée que dans l'année 1874 par Seidel [16, 35]. La méthode de Gauss Seidel suit le même principe que la méthode de Jacobi sauf que Gauss-Seidel bénéficie de la remarque tiré au cours du calcul de $x_i^{(k)}$ et qui dévoile que la nouvelle approximation de $x_j^{(k)}$ pour tout $j < i$ n'est pas mise à profit.

L'expression matricielle de l'algorithme Gauss-Seidel suppose que la matrice A se décompose comme elle est illustrée dans les équations (7.4) et (7.5), en changeant la définition de M et N :

$$M = D - E \text{ et } N = F$$

Alors :

$$(D-E)x = Fx + b \Leftrightarrow x = (D-E)^{-1}(Fx + b) \qquad (7.17)$$

La matrice de Gauss-Seidel est exprimée par le produit $(D-E)^{-1}F$. Pour utiliser le maximum d'information disponible à chaque itération, la méthode de Gauss-Seidel utilise à la place de l'équation (7.14) la récurrence suivante :

$$x^{(k+1)} = D^{-1}(Ex^{(k+1)} + Fx^{(k)} + b) \tag{7.18}$$

Elle devient :

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}}{a_{ii}}, \text{ pour i=1, 2, …, n} \tag{7.19}$$

Alors :

$$\begin{aligned} a_{11}x_1^{(k+1)} &= b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \ldots - a_{1n}x_n^{(k)} \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} &= b_2 - a_{23}x_3^{(k)} - \ldots - a_{1n}x_n^{(k)} \\ \ldots \quad &= \quad \ldots \\ a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + \ldots + a_{nn}x_n^{(k+1)} &= b_n \end{aligned} \tag{7.20}$$

Contrairement à la méthode de Jacobi, Gauss-Seidel ne nécessite qu'un seul vecteur de stockage, à chaque itération la nouvelle valeur calculée $x^{(k+1)}$ et l'ancienne valeur sont stockées dans le même vecteur.

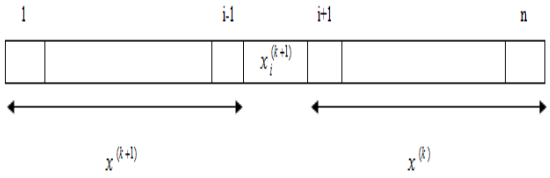

Figure 7.1- Principe itérative de la méthode de Gauss-Seidel

Remarque 7.2 : la dépendance entre les résultats obtenus instantanément et le calcul en cour rend la méthode de Gauss-Seidel difficile à paralléliser.

Nous présentons l'algorithme de Gauss-Seidel:

Algorithme 7.2- Gauss-Seidel

Choix d'une solution initiale $x^{(0)} \in R^n$

Pour chaque k=0, 1, … jusqu'à la convergence faire

Pour chaque i=1, …, n faire

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}}{a_{ii}}$$

Fin pour

Fin pour

Exemple 7.2 : nous calculons les solutions du système illustré dans l'exemple 7.1 en utilisant la méthode de Gauss-Seidel, nous avons les formules suivantes :

$$x_1^{(k+1)} = \frac{7}{10} - \frac{2}{10}x_2^{(k)} + \frac{1}{10}x_3^{(k)}$$
$$x_2^{(k+1)} = -\frac{4}{8} - \frac{1}{8}x_1^{(k+1)} - \frac{3}{8}x_3^{(k)}$$
$$x_3^{(k+1)} = \frac{9}{10} + \frac{2}{10}x_1^{(k+1)} + \frac{1}{10}x_2^{(k+1)}$$

nous débutons par les même solutions initiales $(x_1^{(0)}, x_2^{(0)}, x_2^{(0)}) = (0,0,0)$, nous obtenons:

N	K=0	K=1	K=2	K=…
$x_1^{(k)}$	0	0.7	0.915	
$x_2^{(k)}$	0	-0.587	-0.982	
$x_3^{(k)}$	0	0.981	0.984	

Tableau 7.2- Résultats par itération en utilisant la méthode Gauss-Seidel

La solution exacte du système traité dans l'exemple (7.1) et (7.2) est $(x_1, x_2, x_3) = (1,-1,1)$. En comparant la solution obtenu en utilisant les deux méthodes Gauss-Seidel et Jacobi, nous remarquons que la méthode Gauss-Seidel est celle qui donne les valeurs les plus proches de la valeur exacte au bout da chaque itération, ce qui confirme bien que la méthode de Gauss-Seidel converge plus vite que la méthode de Jacobi.

7.1.3 Convergence de la méthode de Jacobi et de Gauss-Seidel

Le résultat suivant démontre les conditions de convergence de la méthode Jacobi ainsi que la méthode Gauss-Seidel (Voir [54]).

Définition 7.1 : une matrice A est dit à diagonale strictement dominante [62] si les coefficients $|a_{ii}| > \sum_{j \neq i}^{n} |a_{ij}|$ pour tout $i = 1,...,n$.

Théorème 7.1 : si A une matrice à diagonale strictement dominante alors la méthode de Jacobi et Gauss Seidel sont convergentes pour toute approximation initiale $x^{(0)}$.

7.2 Méthodes itératives et Problèmes Décisionnels de Markov

A partir des années 70, le domaine de la recherche dans les PDMs a connu un nouvel axe qui s'intéresse à accélérer la convergence des méthodes itératives utilisées dans leur résolution, en se basant sur le développement des équations de récursivité.

L'algorithme IV (ou IV Pre-Jacobi), calcule itérativement le point fixe de l'opérateur de Bellman L : $LV(s) = \max_{a \in A(s)} \left\{ r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \right\}$ par la formule :

$$V^{t+1} = L(V^t) \text{ pour t=0, 1, …} \tag{7.21}$$

Reprenons le système d'équations linéaires issu de l'équation de Bellman (3.20). Il correspond au système (7.2), avec $x = V_{\pi_k}$, $A = [I - \gamma P(\pi_k)]$ et $b = r(\pi_k)$.

$$[I - \gamma P(\pi_k)]V_{\pi_k} = r(\pi_k) \tag{7.22}$$

On note par :

- $[P(\pi_k)]^D$ la matrice diagonale de $P(\pi_k)$;
- $[P(\pi_k)]^L$ la matrice triangulaire inférieure de $[P(\pi_k)]$de diagonale nulle ;
- $[P(\pi_k)]^U$ la matrice triangulaire supérieure de $[P(\pi_k)]$ de diagonale nulle ;
- $[P(\pi_k)]^{U+L}$ la somme des deux matrices $[P(\pi_k)]^U$ et $[P(\pi_k)]^L$;
- $[P(\pi_k)]^{D+U}$ la somme des deux matrices $[P(\pi_k)]^D$ et $[P(\pi_k)]^U$.

7.2.1 Algorithme d'Itération de la Valeur Jacobi (IV-J)

L'algorithme IV-J développe la version améliorée de IV-PJ, il se base sur la même décomposition (7.9) de la méthode de Jacobi, avec $D = I - \gamma[P(\pi_k)]^D$ et $E + F = \gamma[P(\pi_k)]^L + \gamma[P(\pi_k)]^U = \gamma[P(\pi_k)]^{L+U}$. Ainsi, l'équation (7.22) devient :

$$[I - \gamma[P(\pi_k)]^D - \gamma[P(\pi_k)]^{L+U}]V_{\pi_k} = [r(\pi_k)] \tag{7.23}$$

$$\left[I - \gamma[P(\pi_k)]^D\right]V_{\pi_k}^{t+1} = [r(\pi_k)] + \gamma[P(\pi_k)]^{L+U}V_{\pi_k}^t$$

$$V_{\pi_k}^{t+1} = \frac{[r(\pi_k)] + \gamma[P(\pi_k)]^{L+U}V_{\pi_k}^t}{\left[I - \gamma[P(\pi_k)]^D\right]}$$

Donc, pour tout $s \in S$ la forme finale de l'équation récursive est exprimée sous la forme suivante:

$$V^{t+1}(s) = \max_{a \in A(s)} \left\{ \left(r(s,a) + \gamma \sum_{\substack{s' \in S \\ s' \neq s}} p(s'|s,a)V^t(s') \right) / \left(1 - \gamma p(s|s,a)\right) \right\} \tag{7.24}$$

7.2.2 Algorithme IV Pre-Gauss-Seidel (IV-PGS) et IV Gauss-Seidel (IV-GS)

Entre autres variantes, la version d'algorithme IV qui se base sur la méthode de Gauss-Seidel reste la plus efficace au niveau d'accélération de la convergence. Elle vise à mettre à jour la fonction valeur sauvegardée V^t au fur et à mesure que la nouvelle approximation.

Notons D, E et F, les matrices résultantes après la décomposition de la matrice $[I - \gamma P(\pi_k)]$, alors :

- Pour l'algorithme IV-PGS, la matrice $D-E=I-\gamma[P(\pi_k)]^L$ et la matrice $F=\gamma[P(\pi_k)]^{D+U}$. Ce qui donne le système d'équations suivant:

$$[I-\gamma[P(\pi_k)]^L-\gamma[P(\pi_k)]^{U+D}]V_{\pi_k}=r(\pi_k) \tag{7.25}$$

$$\left[I-\gamma[P(\pi_k)]^L\right]V_{\pi_k}^t=r(\pi_k)+\gamma[P(\pi_k)]^{D+U}V_{\pi_k}^{t-1}$$

$$I\times V_{\pi_k}^t=r(\pi_k)+\gamma[P(\pi_k)]^L V_{\pi_k}^t+\gamma[P(\pi_k)]^{D+U}V_{\pi_k}^{t-1}$$

Pour tout état $s\in S$, la forme finale de l'équation récursive s'écrit :

$$V^t(s)=\max_{a\in A(s)}\left\{r(s,a)+\gamma\sum_{s'<s}p(s'|s,a)V^t(s')+\gamma\sum_{s'\geq s}p(s'|s,a)V^{t-1}(s')\right\} \tag{7.26}$$

L'algorithme 7.3 résume le processus de l'algorithme IV-PGS :

Algorithme 7.3- Algorithme IV-PGS

$t\leftarrow 0$

$V^t(s)=0,\forall s\in S$

Répéter

t ← t+1

Pour chaque $s\in S$

$$V^t(s)=\max_{a\in A(s)}\left\{r(s,a)+\gamma\sum_{1\leq s'<s}p(s'|s,a)V^t(s')+\gamma\sum_{s\leq s'\leq|S|}p(s'|s,a)V^{t-1}(s')\right\} \tag{7.27}$$

Fin pour

Jusqu'à $\max_s\left|V^t(s)-V^{t-1}(s)\right|<\varepsilon$

$$\pi(s)=\arg\max_{a\in A(s)}\left\{r(s,a)+\gamma\sum_{s'\in S}p(s'|s,a)V^{t-1}(s')\right\},\forall s\in S \tag{7.28}$$

Retourner π

- Pour l'algorithme IV-GS, la matrice $D-E=I-\gamma[P(\pi_k)]^D-\gamma[P(\pi_k)]^L$ et la matrice $F=\gamma[P(\pi_k)]^U$, ainsi on a le système d'équations suivant :

$$[I-\gamma[P(\pi_k)]^D-\gamma[P(\pi_k)]^L-\gamma[P(\pi_k)]^U]V_{\pi_k}=[r(\pi_k)]$$

$$\left[I-\gamma[P(\pi_k)]^D-\gamma[P(\pi_k)]^L\right]V_{\pi_k}^{t+1}=[r(\pi_k)]+\gamma[P(\pi_k)]^U V_{\pi_k}^t$$

$$V_{\pi_k}^{t+1}=\frac{[r(\pi_k)]+\gamma[P(\pi_k)]^L V_{\pi_k}^{t+1}+\gamma[P(\pi_k)]^U V_{\pi_k}^t}{\left[I-\gamma[P(\pi_k)]^D\right]}$$

La forme finale de l'équation récursive pour l'algorithme IV-GS pour tout $s\in S$, vérifie :

$$V^t(s) = \max_{a \in A(s)} \left\{ \left(r(s,a) + \gamma \sum_{s'<S} p(s'|s,a) V^t(s') + \gamma \sum_{s'>S} p(s'|s,a) V^{t-1}(s') \right) / \left(1 - \gamma p(s|s,a)\right) \right\} \quad (7.29)$$

7.3 Hiérarchisation et les méthodes itératives

Les algorithmes de résolution des PDMs à base des méthodes itératives telles que Gauss-Seidel et Jacobi n'ont pas bénéficié des avantages de la méthode de décomposition dans le cas des problèmes de grandes tailles. Ainsi, l'objectif de ce paragraphe est de construire un algorithme hiérarchique basé sur la méthode itérative Pré-Gauss-Seidel.

Nous construisons d'une nouvelle façon, par récurrence, les PDMs restreints correspondants à chaque CFC du niveau L_p, p = 0, 1, …, (n-1). Soit $S = \cup\{C_{pk}, p = 0,...,n \text{ et } k \in \{1,...,K(p)\}\}$. Pour $s \in S$, posons $V(s)$ la valeur optimale calculée dans les niveaux précédents. Pour $k \in \{1,2,...,K(p)\}$ le PDM_{pk} représente le PDM restreint à C_{pk} défini par :

- Espace d'états : $S_{pk} = C_{pk}$;
- Espace d'actions : pour $s \in S_{pk}$, $A_{pk}(s) = A(s)$;
- Probabilités de transition : pour $s, s' \in S_{pk}$, $p_{pk}(s'|s,a) = p(s'|s,a)$, $a \in A(s)$;
- Gains : Soit $s \in S_{pk}$, $r_{pk}(s,a) = r(s,a) + \gamma \sum_{s' \in \Gamma(C_{pk})} p(s'|s,a) V(s')$.

7.3.1 Algorithme IV Pré-Gauss-Seidel Hiérarchique (IV-PGS Hiérarchique)

Dans ce paragraphe, nous allons présenter un algorithme amélioré résultant de l'application de la décomposition sur l'algorithme IV-PGS, nommé IV-PGS Hiérarchique. D'après la nouvelle définition des PDMs restreints à chaque CFC, nous pouvons reformuler l'équation d'optimalité (7.26) suivant les niveaux :

- Au niveau L_0, pour tout $s \in C_{0k}$, $k \in \{1,2,...,K(0)\}$:

$$V_{0k}^t(s) = \max_{a \in A_{0k}(s)} \left[r_{0k}(s,a) + \gamma \left[\sum_{\substack{s'<s \\ s' \in C_{0k}}} p_{0k}(s'|s,a) V_{0k}^t(s') + \sum_{\substack{s' \geq s \\ s' \in C_{0k}}} p_{0k}(s'|s,a) V_{0k}^{t-1}(s') \right] \right] \quad (7.30)$$

- Au niveau L_1 : pour tout $s \in C_{1k}$, $k \in \{1,2,...,K(1)\}$:

$$V^{t}(s)=\max_{a\in A(s)}\left[r(s,a)+\gamma\left[\begin{array}{l}\sum_{\substack{s'<s\\ s'\in C_{mk}}}p(s'|s,a)V^{t}(s')+\sum_{\substack{s'\geq s\\ s'\in C_{mk}}}p(s'|s,a)V^{t-1}(s')\\ +\sum_{s'\in\Gamma(C_{mk})}p(s'|s,a)V(s')\end{array}\right]\right]$$

d'après la définition des PDMs restreints dans le niveau L_1, l'espace d'états $S_{1k}=C_{1k}$; l'espace d'actions $A_{1k}(s)=A(s)$, $s\in S_{1k}$; la probabilités de transition $p_{1k}(s'|s,a)=p(s'|s,a)$, pour tout $s,s'\in S_{1k}$, $a\in A(s)$; le gains $r_{1k}(s,a)=r(s,a)+\gamma\sum_{s'\in\Gamma(C_{1k})}p(s'|s,a)V(s')$. Ainsi :

$$V_{1k}^{t}(s)=\max_{a\in A_{1k}(s)}\left[r_{1k}(s,a)+\gamma\left[\sum_{\substack{s'<s\\ s'\in C_{1k}}}p_{1k}(s'|s,a)V_{1k}^{t}(s')+\sum_{\substack{s'\geq s\\ s'\in C_{1k}}}p_{1k}(s'|s,a)V_{1k}^{t-1}(s')\right]\right]$$

- Au niveau L_m , généralement pour tout $m>1, s\in C_{mk}$, $k\in\{1,2,...,K(m)\}$:

$$V_{mk}^{t}(s)=\max_{a\in A_{mk}(s)}\left[r_{mk}(s,a)+\gamma\left[\sum_{\substack{s'<s\\ s'\in C_{mk}}}p_{mk}(s'|s,a)V_{mk}^{t}(s')+\sum_{\substack{s'\geq s\\ s'\in C_{mk}}}p_{mk}(s'|s,a)V_{mk}^{t-1}(s')\right]\right] \quad (7.31)$$

Nous présentons ici le résultat fondamental de ce paragraphe :

Théorème 7.2 : Soit $V_{mh}(s)$, $s\in C_{mh}$ la valeur optimale pour le PDM restreint PDM_{mh}, alors $V_{mh}(s)$ est égale à la valeur optimale $V(s)$ pour le PDM originale.

Preuve : La démonstration est par récurrence. Pour m=0, $k\in\{1,2,\ldots,K(0)\}$, les classes C_{0k} sont fermées. Le vecteur optimal V est la solution unique de (7.32) pour tout $s\in C_{0k}$:

$$V(s)=\max_{a\in A(s)}\left[r(s,a)+\gamma\left[\sum_{\substack{s'<s\\ s'\in C_{0k}}}p(s'|s,a)V(s')+\sum_{\substack{s'\geq s\\ s'\in C_{0k}}}p(s'|s,a)V(s')\right]\right] \quad (7.32)$$

D'après (7.32), à la limite le vecteur V_{0k}^{t} converge au vecteur optimal V_{0k} qui est donc solution de (7.32) :

$$V_{0k}(s)=\max_{a\in A_{0k}(s)}\left[r_{0k}(s,a)+\gamma\left[\sum_{\substack{s'<s\\ s'\in C_{0k}}}p_{0k}(s'|s,a)V_{0k}(s')+\sum_{\substack{s'\geq s\\ s'\in C_{0k}}}p_{0k}(s'|s,a)V_{0k}(s')\right]\right] \quad (7.33)$$

En utilisant les deux équations (7.32), (7.33) et le fait que $A_{0k}(s)=A(s)$, $r_{0k}(s,a)=r(s,a)$ et $p_{0k}(s'|s,a)=p(s'|s,a)$ pour tout $s\in C_{0k}$, il est claire que $V_{0k}(s)=V(s)$.

Soit $m > 0$, supposons que le résultat est vrai pour tout niveau précédent m et nous montrons que le résultat est toujours vrai pour m.

On a V(s), $s \in S_{mk}$ est l'unique solution de l'équation :

$$V(\mathrm{s}) = \max_{a \in A(\mathrm{s})} \left[r(\mathrm{s},a) + \gamma \left[\sum_{\substack{s'<s \\ s' \in C_{mk}}} p(\mathrm{s}'|\mathrm{s},a) V(\mathrm{s}') + \sum_{\substack{s' \geq s \\ s' \in C_{mk}}} p(\mathrm{s}'|\mathrm{s},a) V(\mathrm{s}') + \sum_{s' \in \Gamma(C_{mk})} p(\mathrm{s}'|\mathrm{s},a) V(\mathrm{s}') \right] \right] \quad (7.34)$$

D'après [81] et (7.34) à la limite, le vecteur V_{mk}^{t} converge au vecteur optimal $\mathrm{V_{mk}}$ pour le PDM_{mk} qui est donc solution de :

$$V_{mk}(\mathrm{s}) = \max_{a \in A_{mk}(\mathrm{s})} \left[r_{mk}(\mathrm{s},a) + \gamma \left[\sum_{\substack{s'<s \\ s' \in C_{mk}}} p_{mk}(\mathrm{s}'|\mathrm{s},a) V_{mk}(\mathrm{s}') + \sum_{\substack{s' \geq s \\ s' \in C_{mk}}} p_{mk}(\mathrm{s}'|\mathrm{s},a) V_{mk}(\mathrm{s}') \right] \right] \quad (7.35)$$

En utilisant la définition de $r_{pk}(\mathrm{s},\mathrm{a})$, nous aurons :

$$V_{mk}(\mathrm{s}) = \max_{a \in A_{mk}(\mathrm{s})} \left[r(\mathrm{s},a) + \gamma \left[\sum_{\substack{s'<s \\ s' \in C_{mk}}} p_{mk}(\mathrm{s}'|\mathrm{s},a) V_{mk}(\mathrm{s}') + \sum_{\substack{s' \geq s \\ s' \in C_{mk}}} p_{mk}(\mathrm{s}'|\mathrm{s},a) V_{mk}(\mathrm{s}') + \sum_{s' \in \Gamma(C_{mk})} p(\mathrm{s}'|\mathrm{s},a) V(\mathrm{s}') \right] \right] \quad (7.36)$$

Pour tout $s' \in \Gamma(C_{mk})$ (Avec $\Gamma(C_{mk}) \subset L_{m-1} \cup L_{m-2} \cup ... \cup L_0$) la valeur de V(s') est déjà calculée dans les niveaux précédents, d'après la définition du modèle restreint PDM_{mk}, nous avons:

$$V_{mk}(\mathrm{s}) = \max_{a \in A(\mathrm{s})} \left[r(\mathrm{s},a) + \gamma \left[\sum_{\substack{s'<s \\ s' \in C_{mk}}} p(\mathrm{s}'|\mathrm{s},a) V_{mk}(\mathrm{s}') + \sum_{\substack{s' \geq s \\ s' \in C_{mk}}} p(\mathrm{s}'|\mathrm{s},a) V_{mk}(\mathrm{s}') + \sum_{s' \in \Gamma(C_{mk})} p(\mathrm{s}'|\mathrm{s},a) V(\mathrm{s}') \right] \right] \quad (7.37)$$

Alors V(s), $s \in S_{mk}$ est l'unique solution de l'équation (7.37). Ainsi $\mathrm{V_{mk}}(\mathrm{s})=\mathrm{V}(\mathrm{s})$ pour tout $s \in S_{mk}$.

Corollaire 7.1 : Soit π_{mk} une stratégie optimale pour le PDM_{mk} restreint, alors pour tout $s \in C_{mk}$, $\pi_{mk}(s)$ est une action optimale pour le PDM original.

Preuve : soit p une stratégie optimale, pour tout $s \in S_{mk}$, (D'après le théorème 7.2) nous avons :

$$\pi_{mk}(s) = \arg\max_{a \in A_{mk}(s)} \left[r_{mk}(s,a) + \gamma \sum_{s' \in C_{mk}} P_{mk}(s' \mid s,a) V_{mk}(s') \right] \quad (7.38)$$

$$= \operatorname{Arg}\max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in \Gamma(C_{mk})} P(s' \mid s,a) V(s') + \gamma \sum_{s' \in C_{mk}} P(s' \mid s,a) V(s') \right] = \pi(s) \quad (7.39)$$

Algorithme 7.4- Algorithme IV-PGS Hiérarchique

Pour chaque p=0, …, n-1

Pour chaque k=1, …, K(p)

$t \leftarrow 0$

Pour chaque $s \in S_{pk}$

$V_{pk}^{t}(s) = 0$

Fin pour

Répéter

$t \leftarrow t+1$

Pour chaque $s \in S_{pk}$

$$V_{pk}^{t}(\mathrm{s}) = \max_{a \in A_{pk}(\mathrm{s})} \left[r_{pk}(\mathrm{s},a) + \gamma \left[\begin{array}{l} \sum_{\substack{s'<s \\ s' \in S_{pk}}} p_{pk}(\mathrm{s}' \mid \mathrm{s},a) V_{pk}^{t}(\mathrm{s}') + \\ \sum_{\substack{s' \geq s \\ s' \in S_{pk}}} p_{pk}(\mathrm{s}' \mid \mathrm{s},a) V_{pk}^{t-1}(\mathrm{s}') \end{array} \right] \right]$$

Fin pour

Jusqu'à $\max_{s} | V_{pk}^{t}(s) - V_{pk}^{t-1}(\mathrm{s}) | < \varepsilon$

$$\pi_{pk}(s) = \operatorname{Arg}\max_{a \in A_{pk}(s)} \left\{ r_{pk}(s,a) + \gamma \sum_{s' \in \mathrm{S}_{pk}} p_{pk}(s' \mid s,a) V_{pk}^{t-1}(s') \right\}, \forall s \in S_{pk}$$

Retourner π_{pk}

Fin pour

Fin pour

Selon le même principe, la complexité sera de l'ordre $n\log(n)$ en utilisant le théorème d'Akra Bazzi, cette complexité reste une valeur inférieure à la complexité de l'algorithme IV-Pre-Gauss-Seidel classique (Il a la même complexité que l'algorithme IV).

7.3.2 Résultats expérimentaux

Les expérimentations menées concernent une base générée aléatoirement, la variation du temps de calcul en fonction du nombre des états du système généré et en fonction de l'algorithme utilisé sont résumées dans le tableau suivant :

Etats / Algorithme	500	1000	5000	10000	50000
IV-PGS	0.61	1.82	41.1	165	1144
IV-PGS Hiérarchique	0.04	0.024	0.316	1.7	2.08

Tableau 7.3- Performances des algorithmes IV-PGS et IV-PGS Hiérarchique en fonction du nombre des états du système ($\times 10^{-1} s$)

Dans le tableau 7.3, la première ligne représente le temps résultant d'exécution de l'algorithme IV-PGS (Algorithme 7.3) dans sa forme classique et la deuxième ligne est consacrée au même algorithme mais dans sa version hiérarchique (Algorithme 7.4). Les résultats présentés mettent en évidence l'influence d'augmentation du nombre d'états sur le temps de résolution pour les deux algorithmes. De ce fait, pour un même ensemble d'états, nous remarquons que l'algorithme 7.3 requiert alors plus de temps que l'algorithme 7.4.

Dans la figure 7.2, nous représentons graphiquement l'évolution du temps d'exécution en fonction du nombre des états pour les deux algorithmes.

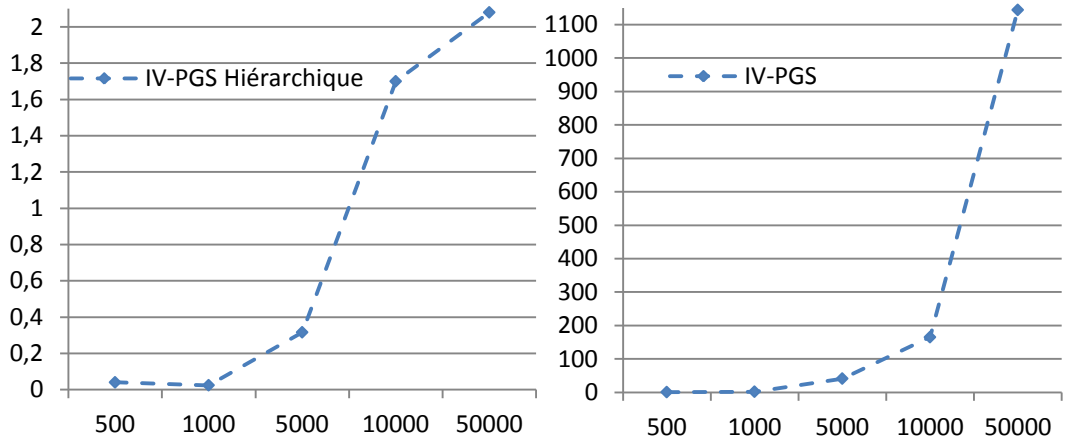

Figure 7.2- Évolution du temps d'exécution pour l'algorithme IV-PGS et IV-PGS Hiérarchique en fonction du nombre d'états ($\times 10^{-1} s$)

L'utilisation de la technique de décomposition, nous a permis de résoudre des modèles de grandes tailles, contrairement aux algorithmes classiques. Les algorithmes de résolution Hiérarchiques, IV-PJ et IV-PGS sont comparés dans le tableau 7.4, Ils peuvent traiter des problèmes qui dépassent 10^5 états.

Tableau 7.4- Performances de l'algorithme IV-PJ Hiérarchique et l'algorithme IV-PGS Hiérarchique (s)

Algorithme \ Etats	10^6	10^7	5×10^7
IV-PJ Hiérarchique	22,5	229	2001,9
IV-PGS Hiérarchique	19,28	193	1584

La comparaison expérimentale citée dans le tableau 7.4, montre que l'algorithme IV-PGS reste toujours plus rapide par apport à l'algorithme IV-PJ même après l'utilisation de la technique de décomposition. Cette comparaison est bien schématisée dans la figure 7.3 pour différents espaces d'états :

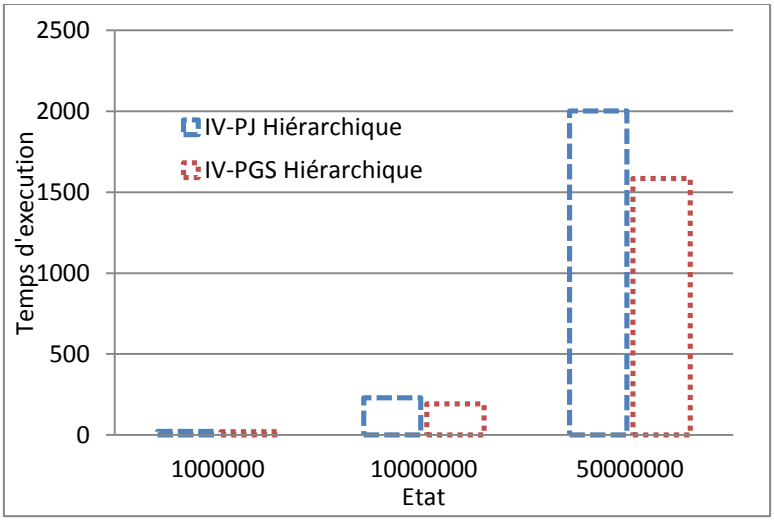

Figure 7.3- Evolution du temps d'exécution pour l'algorithme IV-PJ Hiérarchique et l'algorithme IV-PGS Hiérarchique (s)

En conclusion, l'amélioration que nous avons apportée à l'algorithme IV PGS nous a permis d'obtenir un gain très important au niveau du temps d'exécution, et l'algorithme amélioré reste toujours meilleur que l'algorithme IV-PJ au sens de convergence .

7.4 Conclusion

L'algorithme **IV-PGS Hiérarchique** proposé dans ce chapitre se base sur l'hybridation de la **méthode de décomposition** après l'avoir modifié son principe et la méthode itérative **Pré-Gauss-Seidel**. Cet algorithme est caractérisé par leur efficacité dans les problèmes de grandes tailles et leur convergence rapide vers les solutions optimales en bénéficiant de l'amélioration de l'équation de récursivité introduite par l'utilisation de la méthode de **Gauss-Seidel**.

Dans un premier temps, nous avons détaillé le principe de fonctionnement des méthodes itératives classiques, à savoir, **Jacobi** et Pré-Gauss-Seidel qui sont la base théorique de cet algorithme. Ensuite, nous avons présenté et analysé les résultats expérimentaux et théoriques qui confirment l'efficacité de notre approche en comparant le temps d'exécution pour différentes tailles d'espace d'états. Pratiquement la méthode de Gauss-Seidel converge vers la

solution optimale plus vite que la méthode de Jacobi, Ainsi, à la fin nous avons montré que notre approche garde cette caractéristique en comparant le temps d'exécution de notre algorithme et l'algorithme IV-PJ dont nous avons appliqué la méthode de décomposition.

Nous verrons dans le chapitre suivant qu'une telle représentation des états et telle décomposition seront plus intéressantes en appliquant la technique du **parallélisme**.

8 Parallélisme appliqué aux Problèmes Décisionnels de Markov

Nous avons étudié dans les trois premiers chapitres de cette partie, l'efficacité de la **méthode de décomposition** dans sa version standard ou après les modifications apportées, soit d'une façon directe sur les deux modèles markoviens (Les Problèmes Décisionnels de Markov (PDMs) et les Modèles de Markov Cachés (MMCs)) ou bien dans une forme hybride, en appliquant les méthodes d'accélération de la convergence telles que Gauss-Seidel.

Dans le cas des problèmes de grandes tailles, **le parallélisme** reste l'une des techniques les plus utilisées pour la diminution du temps d'exécution, elle permet aux processeurs de la machine utilisée de fonctionner concurremment pour accroître la puissance de calcul. Ce chapitre, va être consacré au parallélisme hybridé avec des techniques d'améliorations sur les PDMs.

Dans un premier temps, nous allons chercher une stratégie pour paralléliser les algorithmes hiérarchiques (**IV Hiérarchique** et **IV-PGS Hiérarchique**) pour bénéficier des avantages de la technique de décomposition, le paradigme du parallélisme et les méthodes d'amélioration de la convergence. Après, nous allons présenter les algorithmes parallèles résultants de cette approche en montrant leurs complexités.

Ensuite, nous allons proposer une répartition **topologique parallèle** de l'espace d'états du système, ce qui a généré deux algorithmes : l'algorithme d'**IV Topologique Parallèle** et l'algorithme d'**IP Topologique Parallèle** avec un gain considérable en temps de calcul.

8.1 Décomposition et parallélisme

Dans la résolution des PDMs de grandes tailles par des algorithmes classiques du genre itératif tels que l'algorithme d'IV ou l'algorithme d'IV-PGS, chaque itération consiste à résoudre un système linéaire, ce qui provoque un temps d'exécution important. Pour remédier à ce problème, nous avons pensé d'utiliser le paradigme du parallélisme.

Le parallélisme repose sur une bonne stratégie qui génère une indépendance entre les données ce qui permet au processus parallèle d'effectuer son principe. Dans cette section, nous allons bénéficier de l'indépendance naturel des données de la technique de décomposition utilisée pour paralléliser ces deux algorithmes.

D'après notre étude de l'approche de décomposition adoptée, on constate une sorte d'indépendance signalée dans le premier point de la remarque 5.3. En effet, les taches de calcul, soit de la fonction valeur ou bien de la stratégie optimal dans chaque PDM restreint du même niveau, peuvent être exécutées en parallèle. Le passage d'un niveau à un niveau supérieur requiert une dépendance expliquée dans le deuxième point de la remarque 5.3. Le besoin à des valeurs déjà calculées dans les niveaux inferieurs exige un calcul séquentiel entre les niveaux pour que les résultats obtenus dans les niveaux inférieurs soient prises en considération. La figure 8.1 explique le principe de notre approche.

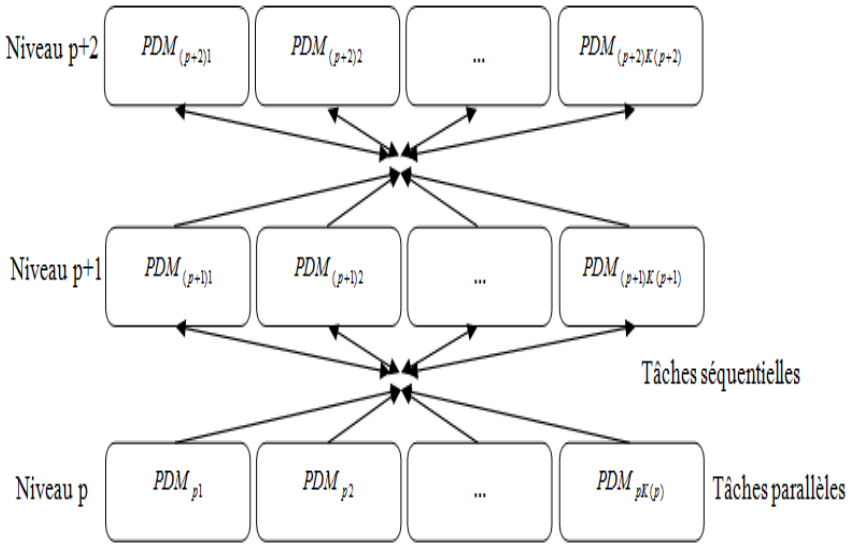

Figure 8.1- Taches parallèles pour les PDMs restreints avec trois niveaux

Nous utilisons une simulation en parallèle sur une machine multiprocesseur à mémoire distribuée. La communication entre les processeurs est faite via l'environnement MPI, cette communication assure la distribution des résultats obtenus pour un niveau avec les autres processeurs pour qu'ils puissent terminer les calculs dans les niveaux supérieurs.

Dans le paragraphe suivant, nous présenterons les résultats expérimentaux de l'implémentation parallèle des deux algorithmes.

8.1.1 Analyse expérimentale de l'algorithme IV Hiérarchique Parallèle

L'algorithme IV Hiérarchique Parallèle est un algorithme résultant de l'approche d'hybridation entre la décomposition et le parallélisme comme il est montré ci-dessous :

Algorithme 8.1- IV Hiérarchique Parallèle

Initialisation et configuration de MPI_Init()

Calculer β_{pi} , le nombre des classes à attribuer pour chaque processeur i dans chaque niveau p

$t \leftarrow 0$

Chaque processeur i exécute la tache suivante :

Pour chaque $s \in C_{pk}$ pour $p = 0,...,(n-1)$ et $k = 1,...,\beta_{pi}$

$V_{pk}^{t}(s) = 0$

Fin pour

Pour chaque niveau p=0, …, (n-1)

Chaque processeur i exécute la tache suivante :

Pour chaque classe k=1,...,β_{pi}

Répéter

t ← t+1

Pour chaque $s \in C_{pk}$

$$V_{pk}^{t}(s) = \max_{a}\left[r_{pk}(s,a) + \gamma \sum_{s' \in S_{pk}} p_{pk}(s' \mid s,a)\, V_{pk}^{t-1}(s') \right] \quad (8.1)$$

$$\pi_{pk}^{t}(s) = \arg\max_{a}\left[r_{pk}(s,a) + \gamma \sum_{s' \in S_{pk}} p_{pk}(s' \mid s,a)\, V_{pk}^{t-1}(s') \right] \quad (8.2)$$

Fin pour

Jusqu'à $\max_{s} | V_{pk}^{t}(s) - V_{pk}^{t-1}(s) | < \varepsilon$

t ← 0

Fin pour

Les données calculées sont échangées par le protocole MPI

Fin pour

Chaque processeur i connait le nombre des CFCs, β_{pi} à traiter dans chaque niveau p, pour p=0, 1, …, (n-1). Nous présentons une analyse détaillée de l'algorithme IV Hiérarchique Parallèle :

Etape 1 : nous débutons par le niveau p=0 à l'instant t=0 ;

Etape 2 : pour tout état $s \in C_{pk}$, chaque processeur i commence avec une valeur arbitraire de la fonction valeur $V_{pk}^{t}(s)$, pour toutes les k classes dont il est sensé à traiter dans le niveau p ;

Etape 3 : en parallèle, pour t={1, 2, …} chaque processeur calcule la nouvelle valeur $V_{pk}^{t+1}(s)$ et la meilleure action pour l'ensemble des états appartenant à C_{pk} suivant les équations (8.1) et (8.2). Chaque processeur peut arrêter le calcul s'il a vérifié la condition d'arrêt ;

Etape 4: chaque processeur est obligé de recueillir les fonctions valeur résultantes (de l'étape 3) des autres processeurs. Puis, il incrémente le niveau p et retourne à l'étape 2 tant que $p \le n-1$;

Etape 5 : le processeur maitre collecte les résultats finals.

Remarque 8.1 : Pour les algorithmes parallèles, la communication entre les processeurs peut absorber tout le gain au niveau de temps de calcul. Dans notre cas, la communication entre processeurs s'articule autour de l'étape 4 et 5 que nous allons étudier dans ce qui suit :

- ***Complexité de l'étape 4*** :

On note par NP le nombre des processeurs utilisés dans l'exécution:

$$O(\sum_{i=0}^{NP}((NP-1)\times m_{0i})+\sum_{i=0}^{NP}((NP-1)\times m_{1i})+...+\sum_{i=0}^{NP}((NP-1)\times m_{ni}))=O((NP-1)\sum_{i=0}^{NP}\sum_{j=0}^{n}m_{ji})$$

$$=O(NP\times|S|) \quad (8.3)$$

Pour chaque niveau p, le temps gaspillé dans la communication est d'ordre $((NP-1)\times m_{pi})$, avec m_{pi} représente le nombre des états dans la totalité des classes dont le $i^{éme}$ processeur est responsable à les traiter, et $\sum_{i=0}^{NP}\sum_{j=0}^{n}m_{ji}$ est égale au nombre |S| des états du système.

- ***Complexité de l'étape 5*** : la collection des données par le processeur maitre représente une opération élémentaire vers chaque processeur : O(NP)

Alors en totalité, le coût de la communication sera d'ordre :

$$O(NP\times|S|)+O(NP)=O(NP\times|S|) \quad (8.4)$$

Le tableau 8.1 montre les résultats obtenus pour un système de taille 1000 états traités sur une machine : Calculateur SGI, de type grappe de calcul, composé de 888 cœurs Xeon cadencés à 2.66 GHz avec 2 Go de mémoire par cœur. L'implémentation est en langage C.

Le premier programme correspond bien au programme séquentiel de l'algorithme d'IV Hiérarchique. Le programme 2 jusqu'à le programme 7 sont exécutés en utilisant respectivement 2, 4, 5, 6, 8 et 10 processeurs.

Programme	Nombre-processeurs/Taille-problème	Temps d'exécution
1	1/1000	1,5841
2	2/1000	0,8
3	4/1000	0,4831
4	5/1000	0,33488
5	6/1000	0,33337
6	8/1000	0,3334
7	10/1000	0,3340

Tableau 8.1- Comparaison du temps d'exécution entre l'algorithme 8.1 et l'algorithme 5.3 (10^{-1} s)

En utilisant le parallélisme, nous remarquons que le temps d'exécution est réduit considérablement, nous avons gagné presque la moitié à partir de la deuxième ligne. La figure 8.2 représente le temps d'exécution résultant pour les sept programmes.

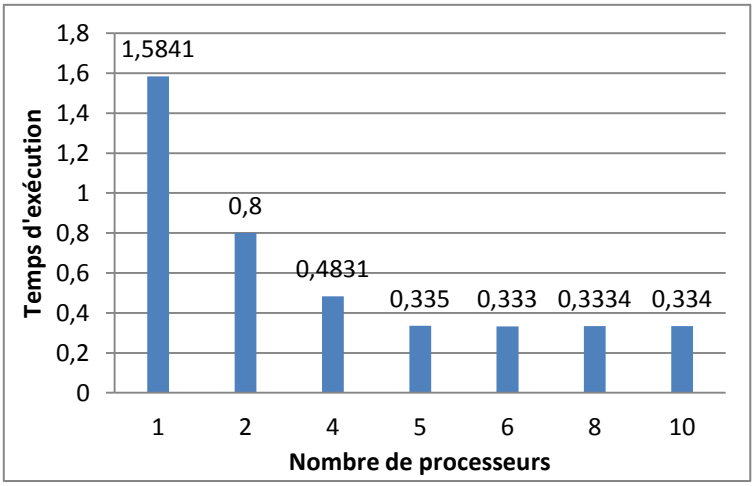

Figure 8.2- Evolution du temps d'exécution pour l'algorithme 8.1 et l'algorithme 5.3 (10^{-1} s)

Dans le tableau 8.2, nous avons calculé l'accélération et l'efficacité correspondantes au temps d'exécution dans le tableau 8.1. Quand le nombre de processeurs augmente la communication entre processeurs à un effet inverse à celui recherché par la parallélisation du programme, ce qui est montré clairement à partir de la sixième ligne du tableau 8.2, En effet, au lieu d'avoir une réduction totale du temps d'exécution et une augmentation d'accélération nous avons trouvé le contraire, ce qui est expliqué par l'accumulation du temps de communication, qui augmente avec le nombre de processeurs.

Programme	Temps d'exécution	Accélération	Efficacité
1	1,5841	--	--
2	0,8	1,9801	0,99
3	0,4831	3,279	0,8197
4	0,33488	4,7303	0,946
5	0,33337	4,7517	0,7919
6	0,3334	4,7513	0,5939
7	0,3340	4,7428	0,4742

Tableau 8.2- Comparaison de l'accélération et de l'efficacité pour l'algorithme 8.1 et l'algorithme 5.3

Dans la figure 8.3, nous montrons l'évolution du temps d'exécution, l'accélération et l'efficacité en fonction du nombre de processeurs utilisés.

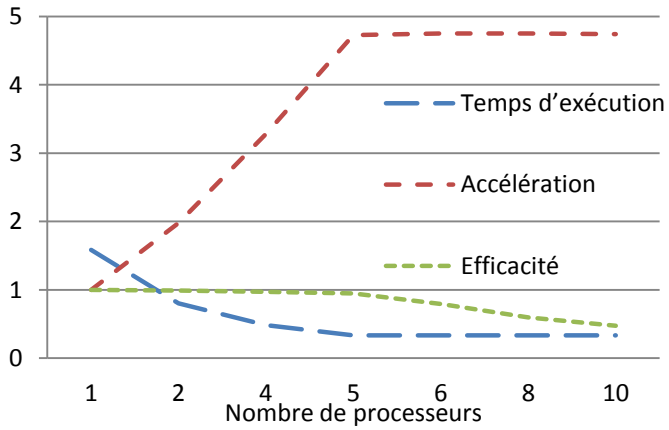

Figure 8.3- Évolution du temps de calcul, l'efficacité et l'accélération en fonction du nombre de processeurs pour l'algorithme 8.1

8.1.2 Analyse expérimentale de l'algorithme IV-PGS Hiérarchique Parallèle

Dans cette section, nous proposons l'algorithme IV-PGS Hiérarchique Parallèle basé sur l'hybridation entre la décomposition adoptée, la méthode d'accélération de la convergence (PGS) et le parallélisme :

Algorithme 8.2- IV-PGS Hiérarchique Parallèle

Initialisation et configuration de MPI_Init()

Calculer β_{pi} , le nombre des classes à attribuer pour chaque processeur i dans chaque niveau p

$t \leftarrow 0$

Chaque processeur i exécute la tache suivante :

Pour chaque $s \in C_{pk}$, p=0, …, (n-1) et k=1, 2, …, β_{pi}

$V_{pk}^{t}(s) = 0$

Fin pour

Pour chaque niveau p=0, …, (n-1)

Chaque processeur i exécute la tache suivante :

Pour chaque classe k=1,...,β_{pi}

Répéter

$t \leftarrow t+1$

Pour chaque $s \in C_{pk}$

$$V_{pk}^{t}(s) = \max_{a}\left[r_{pk}(s,a) + \gamma \left[\begin{array}{l} \sum_{\substack{s'<s \\ s' \in C_{pk}}} p_{pk}(s'|s,a) V_{pk}^{t}(s') + \\ \sum_{\substack{s' \geq s \\ s' \in C_{pk}}} p_{pk}(s'|s,a) V_{pk}^{t-1}(s') \end{array} \right] \right] \quad (8.5)$$

Fin pour

Jusqu'à $\max_s | V_{pk}^t(s) - V_{pk}^{t-1}(s) | < \varepsilon$

$$\pi_{pk}(s) = \arg\max_a \left[r_{pk}(s,a) + \gamma \left[\sum_{s' \in C_{pk}} p_{pk}(s' \mid s,a) V_{pk}^{t-1}(s') + \right] \right] \quad (8.6)$$

$t \leftarrow 0$

Fin pour

Les données calculées sont échangées par le protocole MPI

Fin pour

L'interprétation de l'algorithme 8.2 est similaire à celle de l'algorithme 8.1. Le coût de communication entre processeurs reste le même pour les deux algorithmes $O(NP \times |S|)$ *:*

Etape 1 : nous débutons par le niveau p=0 à l'instant t=0 ;

Etape 2 : pour tout état $s \in C_{pk}$, chaque processeur i commence par une valeur arbitraire de la fonction valeur $V_{pk}^t(s)$, pour toutes les k classes dont il est sensé à traiter dans le niveau p ;

Etape 3 : chaque processeur calcule en parallèle la nouvelle valeur $V_{pk}^t(s)$, t={0, 1, …} pour l'ensemble des états appartenant à C_{pk} en respectant l'équation (8.5) ; il peut arrêter le calcul s'il a vérifié la condition d'arrêt ;

Etape 4: chaque processeur calcule une politique optimale (8.6) puis il partage les fonctions de valeur résultantes (De l'étape 3) avec les autres processeurs. Enfin, il incrémente le niveau p et retourne à l'étape 2 tant que $p \leq (n-1)$;

Etape 5 : le processeur maitre collecte les résultats finals.

Le tableau 8.3 reprend les mêmes paramètres, du tableau 8.2, mesurés pendant l'exécution de l'algorithme IV-PGS Hiérarchique Parallèle et Séquentiel. L'implémentation est en langage C sur une machine : Intel(R) Core(TM) i5-3337U CPU @1.80 GHz 1.80GHz.

	Nombre de processeurs	Temps d'exécution (ms)	Accélération	Efficacité
1000 Etats	1	2 ,4	1	1
	2	1,3	1,84	0,92
	3	0,9	2 ,66	0,88
	4	0,7	3,42	0,85
5000 Etats	1	31,6	1	1
	2	18	1,75	0,87
	3	14,6	2,16	0,72

	4	12	2,63	0,65
10000 Etats	1	170	1	1
	2	87	1,95	0,97
	3	60	2,83	0,943
	4	45	3,77	0,94
50000 Etats	1	2080	1	1
	2	1160	1,79	0,89
	3	800	2,6	0,86
	4	640	3,25	0,81

Tableau 8.3- Comparaison entre l'algorithme IV-PGS Hiérarchique Parallèle et Séquentiel (ms)

Par rapport au programme séquentiel (Un seul processeur) le temps d'exécution des programmes parallèles a diminué considérablement. La figure 8.4, décrit l'évolution du temps de calcul pour des espaces d'états de différentes tailles en utilisant le même nombre de processeurs. Le temps de calcul diminue quand le nombre de processeurs augmentent ce qui est expliqué par l'exécution simultané des taches par la totalité des processeurs.

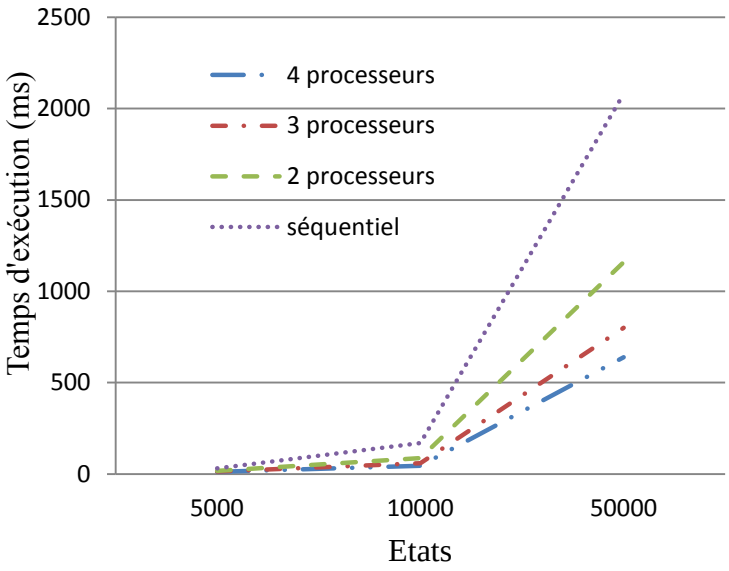

Figure 8.4- Évolution du temps de calcul en fonction de nombre des états pour l'algorithme IV-PGS Hiérarchique et Parallèles

En basant sur une comparaison entre le temps de calcul de l'algorithme IV-PGS Hiérarchique Parallèle et l'algorithme IV-PJ Hiérarchique Parallèle pour des espaces d'états de différentes dimensions, nous pouvons constater que Gauss-Seidel reste plus rapide que Jacobi (Voir le chapitre 3). Le tableau 8.4 illustre cette comparaison :

Algorithmes \ Etats	10^6	10^7	5×10^7
IV-PJ Hiérarchique Parallèle	15,7	138	617
IV-PGS Hiérarchique Parallèle	12	111	500

Tableau 8.4-Influence du parallélisme sur le taux de convergence de l'algorithme IV-PJ Hiérarchique Parallèle et l'algorithme IV-PGS Hiérarchique Parallèle

D'après le tableau 8.4, nous remarquons que le temps consommé par l'algorithme IV-PGS Hiérarchique Parallèle est moins que le temps consommé par l'algorithme IV-PJ Hiérarchique Parallèle. En conclusion, le développement que nous avons apporté aux deux versions de l'algorithme d'IV classique, nous a permis d'obtenir un gain très important au niveau du temps d'exécution, et nous remarquons que, si on combine la méthode de décomposition, les méthodes itératives telles que Gauss-Seidel et Jacobi et le parallélisme on conserve l'ordre d'efficacité entre ces méthodes.

8.2 Parallélisme de la décomposition topologique

L'espace d'état S peut être décomposé en deux sous ensembles suivant un état donné s : $S = \Gamma(s) \cup (S \mid \Gamma(s))$ avec $\Gamma(s)$ est l'ensemble des successeurs de s. Dans l'exemple suivant |S|=8 états (Figure 8.5).

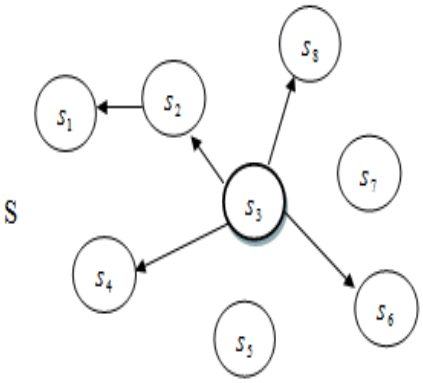

Figure 8.5- Exemple d'un système d'ordre 8

Prenant l'état s_3, alors l'espace d'état est l'union des successeurs de cet état $\Gamma(s_3)$ et le reste des états $(S \mid \Gamma(s_3))$:

$$S = \Gamma(s_3) \cup (S \mid \Gamma(s_3)) \tag{8.7}$$

Avec $\Gamma(s_3) = \{s_2, s_4, s_3, s_6, s_8\}$ et $S \mid \Gamma(s_3) = \{s_1, s_5, s_7\}$.

Nous pouvons bénéficier de cette décomposition de l'espace d'état S, qui se base sur la forme topologique des états du système pour générer une nouvelle version de l'algorithme IV et de l'algorithme IP Amélioré (Algorithme 8.3 et 8.4) dont nous allons appliquer le parallélisme dans les sections suivantes.

L'implémentation sera faite aussi en langage C, avec une machine pareille que celle utilisée dans l'algorithme 8.2, mais au lieu de travailler avec le MPI nous avons choisis l'outil OpenMP qui permet de gérer un programme avec des blocs parallélisées et des blocs qui ne le sont pas. Nous avons abordé ce qu'on appelle la parallélisation des boucles qui se manifeste avec l'ajout des directives de compilation avant les boucles pour avoir une parallélisation automatique.

En fixant l'état étudié s, en utilisant l'équation (8.7) pour reformuler l'équation de mise à jour de la fonction valeur et de la politique pour l'algorithme d'IV.

- La nouvelle formule de la fonction valeur :

$$V^t(s) = \max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V^{t-1}(s') \right] \tag{8.8}$$

$$V^t(s) = \max_{a \in A(s)} \left[r(s,a) + \gamma \left[\sum_{s' \in \Gamma(s)} p(s' \mid s,a) V^{t-1}(s') + \sum_{s' \in (S \mid \Gamma(s))} p(s' \mid s,a) V^{t-1}(s') \right] \right]$$

Pour tout $s' \in (S \mid \Gamma(s))$ la probabilité $p(s' \mid s,a) = 0$, ainsi :

$$V^t(s) = \max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,a) V^{t-1}(s') \right] \tag{8.9}$$

- De même pour la politique :

$$\pi^t(s) = \arg\max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V^{t-1}(s') \right] \tag{8.10}$$

$$\pi^t(s) = \arg\max_{a \in A(s)} \left[r(s,a) + \gamma \left[\sum_{s' \in \Gamma(s)} p(s' \mid s,a) V^{t-1}(s') + \sum_{s' \in (S \mid \Gamma(s))} p(s' \mid s,a) V^{t-1}(s') \right] \right]$$

$$\pi^t(s) = \arg\max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,a) V^{t-1}(s') \right] \tag{8.11}$$

En basant sur les deux équations (8.9) et (8.11), nous proposons l'algorithme IV topologique suivant :

Algorithme 8.3- IV Topologique

$t \leftarrow 0$

Pour tout s $\in S$

$V^0(s) = 0$

Fin pour

Répéter

t $\leftarrow$ t+1

Pour tout $s \in S$ faire

$$V^{t}(s) = \max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,a) V^{t-1}(s') \right] \quad (8.12)$$

$$\pi^{t}(s) = \arg\max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,a) V^{t-1}(s') \right] \quad (8.13)$$

Fin pour

Jusqu'à $\max_{s} | V^{t}(s) - V^{t-1}(s) | < \varepsilon$

Retourner π^{t}

Nous venons d'établir une nouvelle version de l'algorithme d'IV nommée IV Topologique. En appliquant le même principe sur l'algorithme IP Amélioré :

Algorithme 8.4- IP Amélioré Topologique

Seuil de précision ε

$k \leftarrow 0$

Initialisation aléatoire de la politique π_k

Répéter

$t \leftarrow 0$

Pour chaque $s \in S$ faire : $V_{\pi_k}^{t}(s) = 0$

Répéter

Pour chaque $s \in S$ faire

$$V_{\pi_k}^{t+1}(s) = r(s, \pi_k(s)) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s, \pi_k(s)) V_{\pi_k}^{t}(s') \quad (8.14)$$

Fin pour

$t \leftarrow t+1$

Jusqu'à $| V_{\pi_k}^{t}(s) - V_{\pi_k}^{t-1}(s) | < \varepsilon$

Pour chaque $s \in S$ faire

$$\pi_{k+1}(s) = \arg\max_{a \in A(s)} \left[r(s,a) + \gamma \sum_{s' \in \Gamma(s)} p(s' \mid s,a) V_{\pi_k}^{t}(s') \right] \quad (8.15)$$

Fin pour

$k \leftarrow k+1$

Jusqu'à $\pi_k = \pi_{k-1}$

- ***Résultats***

Dans l'algorithme IP amélioré Topologique et l'algorithme IV Topologique (L'algorithme 8.3 et 8.4) nous ne faisons appel qu'aux états successeurs de l'état s pendant la mise à jour de la fonction valeur V et la politique π correspondantes à cet état, ce qui permet de diminuer leur complexité en nombre d'états. Il y'a une transition de l'état s vers un état s' si la probabilité de passage de s vers s' est non nulle $p(s'|s) \neq 0$. On calcule le nombre de transition en sommant tous les probabilités non nulles : $\sum_{j=1}^{|S|} \sum_{\substack{i=1 \\ p(i|j)>0}}^{|S|} 1 = |S| \times K$.

Alors la complexité de nos algorithmes est en ordre $|S| \times K$. Avec K est le nombre moyen de transitions dans le système, qui est en effet une valeur très petite vu le nombre d'états |S|. Ainsi, la complexité des algorithmes proposés est moins que la complexité des algorithmes classiques en nombre d'état.

Passant maintenant à l'étude expérimentale, nous avons calculé le temps résultant après l'exécution de l'algorithme 3.1 séquentiel et l'algorithme 8.3 Parallèle pour des programmes de taille respectif 10^3, 10^4, 5×10^4 et 10^5 états, avec un moyen de successeurs égale à la moitié du nombre d'états utilisés. Les résultats que nous avons obtenus sont représentés dans la figure 8.6.

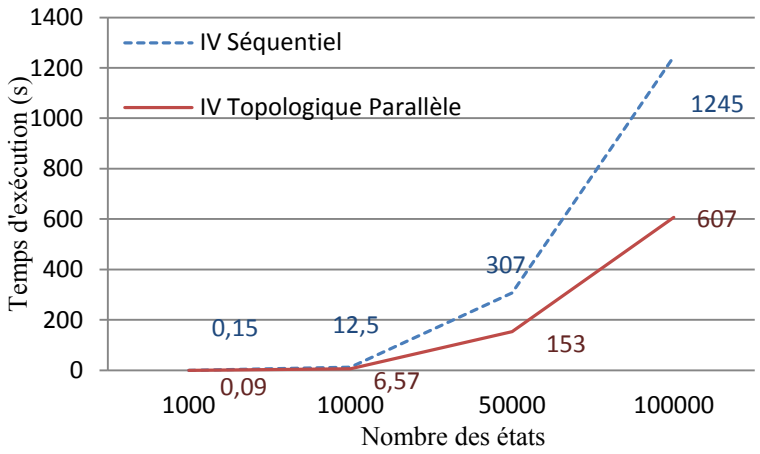

Figure 8.6 - Évolution du temps d'exécution en fonction du nombre d'états pour les deux algorithmes IV Topologique Parallèle et IV Séquentiel (s)

Les résultats présentés dans la figure 8.6, mettent en évidence l'influence du nombre d'états sur le temps de résolution. Nous pouvons, constater que le fait d'augmenter ce nombre provoque une augmentation du temps d'exécution pour les deux algorithmes, mais l'algorithme parallèle reste meilleur. Nos résultats expérimentaux confirment que la meilleure performance est atteinte par l'utilisation de nos approches.

La figure 8.7 exprime l'évolution de temps de calcul en fonction du nombre moyen de successeurs en utilisant l'algorithme 8.3 Parallèle. L'exécution sera appliquée sur un système de dimension 10^7 états et qui absorbe des heures pour arriver à la solution si nous utilisons l'algorithme séquentiel.

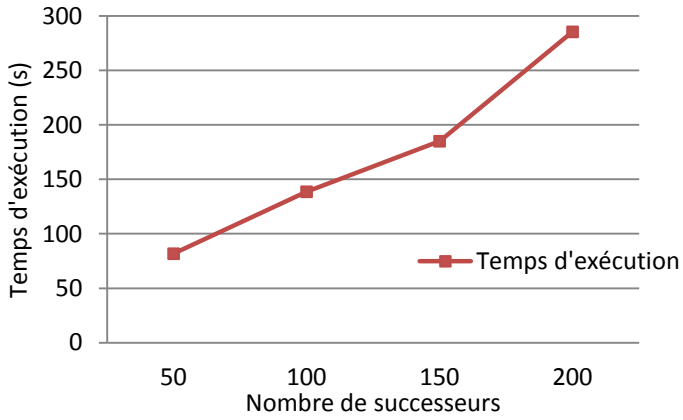

Figure 8.7- Évolution de temps d'exécution d'un modèle de taille 10^7 en fonction du nombre moyen de successeurs pour chaque état (s)

L'augmentation du nombre de successeurs de 50 à 200 pour chaque état a presque quadruplé le temps de calcul. En effet, l'augmentation du nombre de successeurs influence inversement sur l'efficacité de l'approche, mais cet algorithme parallélisé reste toujours meilleur que l'algorithme séquentiel.

Nous finalisons par la validation de l'approche pour l'algorithme 8.4 Parallèle, la figure 8.8 fait varier le nombre d'états du système en calculant le temps consommé pour les deux algorithmes IP Amélioré Séquentiel et IP Amélioré Topologique Parallèle.

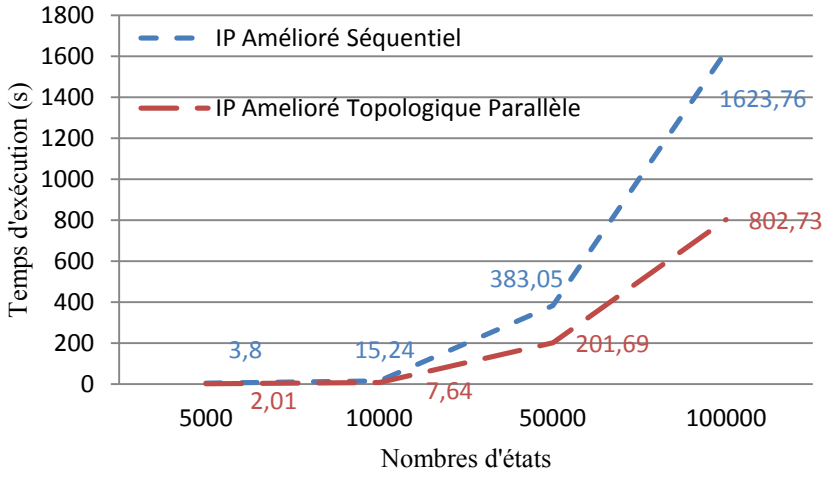

Figure 8.8- Évolution du temps d'exécution en fonction du nombre d'états pour les deux algorithmes IP Amélioré Topologique Parallèle et IP Amélioré Séquentiel (s)

Nous constatons que l'algorithme 8.4 Parallèle est beaucoup plus rapide que l'algorithme séquentiel pour les différents programmes.

8.3 Conclusion

Dans ce chapitre, nous nous sommes intéressés essentiellement aux apports du **parallélisme,** sur les Problèmes Décisionnels de Markov de **grandes tailles**, ce paradigme permet d'utiliser plusieurs processeurs fonctionnant simultanément pour accroître la puissance

de calcul. Nous avons tout d'abord présenté notre approche qui combine **la méthode de décomposition** , la méthode d'accélération de convergence **Gauss-Seidel** et le **parallélisme distribué**. Nous avons explicité les algorithmes parallèles résultants de cette approche qui nous a permet d'avoir des algorithmes plus efficace au niveau de temps d'exécution de convergence et la complexité et nous a permet aussi de paralléliser des algorithmes qui sont naturellement difficile à paralléliser tel que IV-PRG. Ensuite, en décomposant l'espace d'états de façon topologique, nous avons proposé deux algorithmes **IV et IP Topologique** que nous les avons parallélisé par la suite, ce qui nous a donné un gain considérable en temps de calcul. La dernière section de ce chapitre comporte une analyse détaillée des résultats des expérimentales menées avec une étude de complexité des différents algorithmes proposés validant nos approches.

Conclusion de la partie II

Cette partie a porté sur la définition, la validation et l'expérimentation de nos approches pour les **Problèmes Décisionnels de Markov** ainsi que pour les **Modèles de Markov Cachés**.

Comme la méthode de décomposition adoptée ni applicable que pour les algorithmes classique de résolution des Problèmes Décisionnels de Markov tels que l'algorithme **d'Itération de la Valeur** et l'algorithme d'**Itération de la Politique**, nous avons tout d'abord modifié cette **technique de décomposition** pour l'appliquer aux Modèles de Markov Cachés et de l'hybrider avec une méthode d'accélération de convergence de type **Gauss-Seidel**. Ceci nous a permis de proposer **des algorithmes de type hiérarchique** plus intéressants au niveau de temps d'exécution, temps de convergence et la complexité.

Par la suite nous avons traité l'apport du **parallélisme** sur la méthode de décomposition utilisée ainsi que sur une nouvelle décomposition topologique proposée. Ce genre d'hybridation a produit des nouveaux algorithmes **hiérarchiques, topologiques et parallèles**. Leurs validations et efficacités sont vérifiées par l'étude de la complexité et par la comparaison du temps de calcul résultant.

Conclusion générale et perspectives

Conclusion générale

Dans cette thèse, nous avons étudié les Modèles Markoviens de grandes tailles et plus particulièrement les Problèmes Décisionnels de Markov et les Modèles de Markov Cachés.

Nous avons tout d'abord étudié la méthode de décomposition utilisée dans le cas des Problèmes Décisionnels de Markov, et qui représente une solution pour la problématique des espaces d'états de grandes dimensions, elle est confinée dans les algorithmes classiques tels que l'algorithme d'Itération de la Valeur et l'algorithme d'Itération de la Politique. Les approches que nous avons proposées visent à modifier cette technique de décompositions pour l'appliquer aux Modèles de Markov Cachés, et l'hybrider avec une méthode d'accélération de convergence Gauss-Seidel introduite dans le domaine des Problèmes Décisionnels de Markov. Ce qui nous a permis d'introduire de nouveaux algorithmes de type hiérarchique plus efficaces que les versions classiques.

La deuxième contribution se base sur l'application du parallélisme, vu son importance dans l'accélération du temps de calcul ainsi que son efficacité dans les problèmes de grandes tailles. Dans un premier temps, nous avons présenté une approche qui combine la décomposition, le parallélisme distribué et la méthode d'accélération de convergence Gauss-Seidel qui n'est pas parallélisable par nature. Nous avons aussi présenté les algorithmes parallèles résultants de cette approche. Dans un second temps, nous avons hybridé la technique de parallélisme avec une technique de décomposition topologique que nous avons développée, ainsi, nous avons proposé quatre algorithmes : Itération de la Valeur et Itération de la Politique, Topologiques et Parallèles, ils sont caractérisés par un considérable gain dans le temps de calcul par rapport aux algorithmes classiques.

L'étude de complexité des algorithmes améliorés ainsi que les expérimentations menées lors de la validation de nos approches, nous ont permis de démontrer l'adéquation entre notre objectif et les solutions que nous avons proposées.

En conclusion, les algorithmes développés dans le cadre de cette thèse nous offrent un moyen efficace pour résoudre des problèmes réels de grandes tailles modélisés par les Problèmes Décisionnels de Markov ou les Modèles de Markov Cachés.

Perspectives

- *Méthodologiques*

Plusieurs perspectives à la partie méthodologique de la thèse sont envisageables et qui sont relatives à la complication des modèles. Premièrement, nous avons travaillé avec des grands espaces d'états, alors nous pourrions étendre nos approches à la problématique des espaces d'actions de grandes tailles.

Le critère d'actualisation est le seul critère considéré durant notre étude. Alors, nous pourrions également étendre nos approches au cas des autres critères tels que le critère de la moyenne ou le critère pondéré.

- *Appliquées*

Dans la cote applicative, nous envisagerons valider et appliquer nos approches sur un domaine réel et plus précisément en Robotique.

Bibliographie

[1] Abbad, M., et Boustique, H. (2003). A decomposition algorithm for limiting average Markov decision problems. Operations Research Letters, 31(6), 473-476.

[2] Abbad, M., et Daoui, C. (2003). Hierarchical algorithms for discounted and weighted Markov decision processes. Mathematical Methods of Operations Research, 58(2), 237-245.

[3] Akra, M., et Bazzi, L. (1998). On the solution of linear recurrence equations. Computational Optimization and Applications, 10(2), 195-210.

[4] Asadian, A., Kermani, M. R., et Patel, R. V. (2010, June). Accelerated needle steering using partitioned value iteration. In American Control Conference (ACC), 2010 (pp. 2785-2790). IEEE.

[5] Avsar, Z. M., et Baykal-Gürsoy, M. (1999). A decomposition approach for undiscounted two-person zero-sum stochastic games. Mathematical Methods of Operations Research, 49(3), 483-500.

[6] Batcher, K. (1980) Design of a Massively parallel processor. IEEE Transactions on Computers C-29: 836±840

[7] Bather, J. (1973). Optimal decision procedures for finite Markov chains. Part III: General convex systems. Advances in Applied Probability, 541-553.

[8] Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. The annals of mathematical statistics, 41(1), 164-171.

[9] Bellman, R (1957). Dynamic Programming.

[10] Bellman, R. (1961). Adaptive control processes: a guided tour Princeton University Press. Princeton, New Jersey, USA.

[11] Ben Amara N., Belaïd A., Ellouze N., "Utilisation des modèles markoviens en reconnaissance de l'écriture arabe état de l'art," Colloque International Francophone sur l'Ecrit et le Document (CIFED'00), Lyon, France, 2000.

[12] Berrani, S. A. (2004). Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision; application à la recherche d'images par le contenu (Doctoral dissertation, Université Rennes 1).

[13] Bershad, B. N., et Zekauskas, M. J. (1991). Midway: Shared memory parallel programming with entry consistency for distributed memory multiprocessors.

[14] Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999, January). When is “nearest neighbor” meaningful?. In International conference on database theory (pp. 217-235). Springer Berlin Heidelberg.

[15] Beynier, A., et Mouaddib, A. I. (2004). Processus décisionnels de markov décentralisés et interactifs avec contraintes temporelles. Journal Eléctronique d'Intelligece Artificielle (JEDAI).

[16] Black, N., & Moore, S. (2006). Gauss-seidel method. From MathWorld-A Wolfram Web Resource, 1, 1.

[17] Blackwell, D. (1965). Discounted dynamic programming. The Annals of Mathematical Statistics, 36(1), 226-235.

[18] Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., & Mihailidis, A. (2005, July). A decision-theoretic approach to task assistance for persons with dementia. In IJCAI (pp. 1293-1299).

[19] Boutilier, C., Dean, T., et Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. Journal of Artificial Intelligence Research, 11(1), 94.

[20] Boutilier, C., Dearden, R., et Goldszmidt, M. (1995, August). Exploiting structure in policy construction. In IJCAI (Vol. 14, pp. 1104-1113).

[21] Boutilier, C., Dearden, R., et Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. Artificial Intelligence, 121(1), 49-107.

[22] Buongiorno, J., et Zhou, M. (2004). The use of Markov optimization models in the economic and ecological management of forested landscapes under risk.

[23] Bouyssou, D. (2000). Evaluation and decision models: a critical perspective (Vol. 32). Springer Science et Business Media.

[24] Caucal, D. (1990). Graphes canoniques de graphes algébriques. Informatique théorique et Applications, 24(4), 339-352.

[25] Chanel, C. P. C., Teichteil-Königsbuch, F., & Lesire, C. (2013, July). Multi-Target Detection and Recognition by UAVs Using Online POMDPs. In AAAI.

[26] Chen, P., et Lu, L. (2013, October). Markov Decision Process Parallel Value Iteration Algorithm On GPU. In 2013 International Conference on Information Science and Computer Applications (ISCA 2013). Atlantis Press.

[27] Daoui, C., & Abbad, M. (2007). On some algorithms for limiting average Markov decision processes. Operations Research Letters, 35(2), 261-266.

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

[29] Dagum, L., et Enon, R. (1998). OpenMP: an industry standard API for shared-memory programming. Computational Science et Engineering, IEEE, 5(1), 46-55.

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

[31] Denardo, E. V., et Fox, B. L. (1968). Multichain Markov renewal programs. SIAM Journal on Applied Mathematics, 16(3), 468-487.

[32] Drmota, M., et Szpankowski, W. (2013). A master theorem for discrete divide and conquer recurrences. Journal of the ACM (JACM), 60(3), 16.

[33] Duncan, R. (1990). A survey of parallel computer architectures. Computer, 23(2), 5-16.

[34] El Ghazi, A., & Daoui, C. (2015). Phonemes Classification Using the Spectrum. Indonesian Journal of Electrical Engineering and Computer Science, 15(2), 368-372.

[35] El Kourd, K., Naoul, A., & Malki, A. (2015). Comparison from Accuracy Time between Artificial Neural Networks (ANN) of Matlab & Iterative Resolution with Gauss Seidal to

Detect the Lesion. American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS), 14(2), 53-64.

[36] Federgruen, A., et Schweitzer, P. J. (1979). A survey of asymptotic value-iteration for undiscounted Markovian decision processes.

[37] Flynn, M. J. (1966). Very high-speed computing systems. Proceedings of the IEEE, 54(12), 1901-1909.

[38] Flynn, M. J. (1972). Some computer organizations and their effectiveness. Computers, IEEE Transactions on, 100(9), 948-960.

[39] Fortz, B. INFO-F-203: Algorithmique 2.

[40] Frankel, S. P. (1950). Convergence rates of iterative treatments of partial differential equations. Mathematical Tables and Other Aids to Computation, 4(30), 65-75.

[41] Elnashar, A. I. (2011). Parallel performance of MPI sorting algorithms on dual-core processor windows-based systems. arXiv preprint arXiv:1105.6040.

[42] Garet ,O. (2007). Introduction aux Graphes, universite de nancy. Repéré dans: http://iecl.univ-lorraine.fr/

[43] Gondran, M., et Minoux, M. (1995). Graphes et algorithmes.

[44] Gondran, M. (1975). Algèbre linéaire et Cheminement dans un Graphe. Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, 9(1), 77-99.

[45] Gropp, W., Lusk, E., et Skjellum, A (1999). Using MPI: portable parallel programming with the message-passing interface (Vol. 1). MIT press.

[46] Gropp, W. (2002). MPICH2: A new start for MPI implementations. In Recent Advances in Parallel Virtual Machine and Message Passing Interface (pp. 7-7). Springer Berlin Heidelberg.

[47] Guestrin, C., Koller, D., Parr, R., et Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. Journal of Artificial Intelligence Research, 399-468.

[48] Guestrin, C., & Gordon, G. (2002, August). Distributed planning in hierarchical factored MDPs. In Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence (pp. 197-206). Morgan Kaufmann Publishers Inc..

[49] Hamila, M. A. (2012). Planification multi-agents dans un cadre markovien: les jeux stochastiques à somme générale (Doctoral dissertation, Université de Valenciennes et du Hainaut-Cambresis).

[50] Hélary, J. M. (2004). Algorithmique des graphes. Rennes university publication.

[51] Herzberg, M., and U. Yechiali. Accelerated Procedures of The Value Iteration Algorithm For Discounted Markov Decision Processes, Based on One-Step Lookahead Analysis. Operations Research, 42:940–946, 1994.

[52] HOWARD R. (1960). Dynamic Programming and Markov Processes , Wiley, New York.

[53] Howard, R. A. (1963). SEMI-MARKOVIAN DECISION-PROCESSES. Bulletin of the International Statistical Institute, 40(2), 625-652.

[54] James, K. R. (1973). Convergence of matrix iterations subject to diagonal dominance. SIAM Journal on Numerical Analysis, 10(3), 478-484.

[55] JP Leclercq (2010). Cours de Théorie des Graphes et réseaux de Petri

[56] Jin, X., Tan, H. H., et Sun, J. (2007). A STATE-SPACE PARTITIONING METHOD FOR PRICING HIGH - DIMENSIONAL AMERICAN - STYLE OPTIONS. Mathematical Finance, 17(3), 399-426.

[57] Juang B. H.; Rabiner L. R., "Hidden Markov Models for Speech Recog

[58] Kallenberg, L. C. M. (2002). Classification problems in MDPs. In Markov processes and controlled Markov chains (pp. 151-165). Springer US.

[59] Kok, J. R. (2006). Coordination and learning in cooperative multiagent systems. Universiteit van Amsterdam [Host].

[60] Krogh, A., Mian, I. S., et Haussler, D. (1994). A hidden Markov model that finds genes in E. coli DNA. Nucleic Acids Research, 22(22), 4768-4778.

[61] Kushner, H. J. (1971). Introduction to stochastic control. New York: Holt, Rinehart and Winston.

[62] Lemaire, B. (1985). Régularité des matrices à diagonale dominante. Applications à l'absorption dans les chaînes et processus de Markov. Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, 19(3), 233-241.

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

[64] Lozenguez, G., Adouane, L., Beynier, A., Martinet, P., et Mouaddib, A. I. (2013). Résolution approchée par décomposition de processus décisionnels de Markov appliquée à l'exploration en robotique. In 8èmes Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes, JFPDA 2013.

[65] MacQueen. J. (1967). "A test for sub-optimal actions in Markovian decision problems", Operations Res. 15, 559-561

[66] Meuleau, N., Hauskrecht, M., Kim, K. E., Peshkin, L., Kaelbling, L. P., Dean, T. L., et Boutilier, C. (1998, July). Solving very large weakly coupled Markov decision processes. In AAAI/IAAI (pp. 165-172).

[67] Miellou, J. C., & Spitéri, P. (1985). Un critère de convergence pour des méthodes générales de point fixe. RAIRO-Modélisation mathématique et analyse numérique, 19(4), 645-669.

[68] Müller, D. (2012). Introduction à la théorie des graphes. Commission romande de mathématique.

[69] Nakai M., Akira N., Shimodaira H., Sagayama S., "Substroke Approach to HMM-based On-line Kanji Handwriting Recognition," Sixth International Conference on Document Analysis and Recognition (ICDAR 2001). pp.491-495. 2001.

[70] Osaki, S., et Mine, H. (1968). Linear programming algorithms for semi-Markovian decision processes. Journal of Mathematical Analysis and Applications, 22(2), 356-381.

[71] Pellegrini, T., et Duée, R. (2003). Suivi de Voix Parlée grace aux Modèles de Markov Cachés. lieu: IRCAM, 1.

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

[73] Parr, R. (1998, July). Flexible decomposition algorithms for weakly coupled Markov decision problems. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (pp. 422-430). Morgan Kaufmann Publishers Inc..

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

[75] PDMIA, G. (2008). Processus décisionnels de Markov en intelligence artificielle. Edité par Olivier Buffet et Olivier Sigaud, 1. (Cité en page 41.)

[76] Polaka, I., & Borisov, A. Robust dimensionality reduction in bioinformatics data. In 21st European Meeting on Cybernetics and Systems Research (EMCSR 2012): Book of Abstracts (pp. 286-289).

[77] Poļaka, I., & Borisovs, A. (2011). Impact of Feature Selection on Classifier Testing Validity. publication. editionName, 411-418.

[78] Porter, R., Nudelman, E. et Shoham, Y. (2008). Simple search methods for finding a Nash equilibrium. Games and Economic Behavior, 63(2):642–662.

[79] Porteus, E. L. (1975). Bounds and transformations for discounted finite Markov decision chains. Operations Research, 23(4), 761-784.

[80] Porteus, E. L. (1971). Some bounds for discounted sequential decision processes. Management Science, 18(1), 7-11.

[81] Puterman, M. L. (1994). Markov decision processes: discrete stochastic dynamic programming. John Wiley et Sons.

[82] Puterman, M. L., et Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. Management Science, 24(11), 1127-1137.

[83] Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2), 257-286.

[84] Radoszycki, J. (2015). Résolution de processus décisionnels de Markov à espace d'état et d'action factorisés-Application en agroécologie (Doctoral dissertation, INSA de Toulouse). (Cité en page 25.)

[85] Ramy Al-Hajj M., Mokbel C., Likforman-Sulem L., "Reconnaissance de l’écriture arabe cursive: combinaison de classifieurs MMCs à fenêtres orientées," Université de Balamand, Faculté de Génie, pp 1-6. 2006.

[86] Reetz, D. (1977). Approximate solutions of a discounted Markovian decision process. Bonner Mathematische Schriften, 98, 77-92.

[87] Régin, J., Malapert A. (2005). Théorie des Graphes.

[88] Riley, J. D. (1954). Iteration procedures for the Dirichlet difference problem.Mathematical Tables and Other Aids to Computation, 8(47), 125-131.

[89] Ross, K. W., et Varadarajan, R. (1991). Multichain Markov decision processes with a sample path constraint: a decomposition approach. Mathematics of Operations Research, 16(1), 195-207.

[90] Rossi, F., et Villa-Vialaneix, N. (2012). Représentation d’un grand réseau à partir d’une classification hiérarchique de ses sommets. Journal de la Société Française de Statistique, 152(3), 34-65.

[91] Roura, S. (2001). Improved master theorems for divide-and-conquer recurrences. Journal of the ACM (JACM), 48(2), 170-205.

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

[93] Spaan, M. T., et Spaan, M. T. (2004, January). A point-based POMDP algorithm for robot planning. In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on (Vol. 3, pp. 2399-2404). IEEE.

[94] Singh, S., et Cohn, D. (1998). How to dynamically merge Markov decision processes. Advances in neural information processing systems, (10), 1057-1063.

[95] Stéphane, M. Big data cherche mathématiques - Notes du cours. école Normale Supérieure, SMAI. Repéré dans l'environnement mathsmonde: http://mathsmonde.math.cnrs.fr/images/pdf/part2/1-BigDataSmai.pdf

[96] Tarjan, R. (1972). Depth-first search and linear graph algorithms. SIAM journal on computing, 1(2), 146-160.

[97] Tomesco, I. (1967). Méthode pour la détermination de la fermeture transitive d'un graphe fini. Revue française d'informatique et de recherche opérationnelle, 1(4), 33-37.

[98] Thinking Machines Corporation, Cambridge, Ma. (1991) Connection Machine CM-200 Series - Technical Summary.

[99] Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. J. of Math, 58(345-363), 5.

[100] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. Information Theory, IEEE Transactions on, 13(2), 260-269.

[101] Wingate, D., et Seppi, K. D. (2003). Efficient Value Iteration Using Partitioned Models. In ICMLA (pp. 53-59).

[102] Wingate, D., et Seppi, K. D. (2004). P3VI: A partitioned, prioritized, parallel value iterator. In Proceedings of the twenty-first international conference on Machine learning (p. 109). ACM.

[103] Wingate, D., et Seppi, K. D. (2005). Prioritization methods for accelerating MDP solvers. In Journal of Machine Learning Research (pp. 851-881).

[104] Young, D. (1954). Iterative methods for solving partial difference equations of elliptic type. Transactions of the American Mathematical Society, 76(1), 92-111.

[105] Young, D. M. (1972). Second-degree iterative methods for the solution of large linear systems. Journal of Approximation Theory, 5(2), 137-148.

[106] Zang, P., Irani, A., et Isbell Jr, C. L. (2007). Horizon-based value iteration.

[107] Zhang, Q., Sun, G., et Xu, Y. (09). Parallel Algorithms for Solving Markov Decision Process. In Algorithms and Architectures for Parallel Processing (pp. 466-477). Springer Berlin Heidelberg.