Dynamique d'apprentissage pour Monte Carlo Tree Search : applications aux jeux de Go et du Clobber solitaire impartial

André Fabbri

Université de Lyon - École doctorale InfoMaths

Dynamique d'apprentissage pour Monte Carlo Tree Search

Applications aux jeux de Go et du Clobber solitaire impartial

par André Fabbri

Manuscrit pour l'obtention du grade de docteur en informatique

Thèse soutenue le 22 octobre 2015 devant le jury composé de

Rapporteurs :

M. Tristan Cazenave	Professeur	à l'Université Paris-Dauphine - LAMSADE
M. Alain Dutech	Chargé de recherche	à l'INRIA - LORIA
M. Rita Maria Silva Julia	Professeure	à l'Universidade Federal de Uberlândia

Examinateurs :

Mme. Marie-Pierre Gleizes	Professeure	à l'Université Toulouse 3 Paul Sabatier - IRIT
M. Philippe Mathieu	Professeur	à l'Université Lille 1- LIFL

Encadrants :

Mme. Salima Hassas	Professeure	à l'Université Claude Bernard Lyon 1- LIRIS
M. Frédéric Armetta	Maître de Conférences	à l'Université Claude Bernard Lyon 1- LIRIS
M. Éric Duchêne	Maître de Conférences	à l'Université Claude Bernard Lyon 1- LIRIS

UNIVERSITE CLAUDE BERNARD - LYON 1

Président de l'Université	**M. François-Noël GILLY**
Vice-président du Conseil d'Administration	M. le Professeur Hamda BEN HADID
Vice-président du Conseil des Etudes et de la Vie Universitaire	M. le Professeur Philippe LALLE
Vice-président du Conseil Scientifique	M. le Professeur Germain GILLET
Directeur Général des Services	M. Alain HELLEU

COMPOSANTES SANTE

Faculté de Médecine Lyon Est – Claude Bernard	Directeur : M. le Professeur J. ETIENNE
Faculté de Médecine et de Maïeutique Lyon Sud – Charles Mérieux	Directeur : Mme la Professeure C. BURILLON
Faculté d'Odontologie	Directeur : M. le Professeur D. BOURGEOIS
Institut des Sciences Pharmaceutiques et Biologiques	Directeur : Mme la Professeure C. VINCIGUERRA
Institut des Sciences et Techniques de la Réadaptation	Directeur : M. le Professeur Y. MATILLON
Département de formation et Centre de Recherche en Biologie Humaine	Directeur : Mme. la Professeure A-M. SCHOTT

COMPOSANTES ET DEPARTEMENTS DE SCIENCES ET TECHNOLOGIE

Faculté des Sciences et Technologies	Directeur : M. F. DE MARCHI
Département Biologie	Directeur : M. le Professeur F. FLEURY
Département Chimie Biochimie	Directeur : Mme Caroline FELIX
Département GEP	Directeur : M. Hassan HAMMOURI
Département Informatique	Directeur : M. le Professeur S. AKKOUCHE
Département Mathématiques	Directeur : M. le Professeur Georges TOMANOV
Département Mécanique	Directeur : M. le Professeur H. BEN HADID
Département Physique	Directeur : M. Jean-Claude PLENET
UFR Sciences et Techniques des Activités Physiques et Sportives	Directeur : M. Y.VANPOULLE
Observatoire des Sciences de l'Univers de Lyon	Directeur : M. B. GUIDERDONI
Polytech Lyon	Directeur : M. P. FOURNIER
Ecole Supérieure de Chimie Physique Electronique	Directeur : M. G. PIGNAULT
Institut Universitaire de Technologie de Lyon 1	Directeur : M. le Professeur C. VITON
Ecole Supérieure du Professorat et de l'Education	Directeur : M. le Professeur A. MOUGNIOTTE
Institut de Science Financière et d'Assurances	Directeur : M. N. LEBOISNE

Titre

Dynamique d'apprentissage pour *Monte Carlo Tree Search* : applications aux jeux de Go et du Clobber solitaire impartial.

Résumé

Depuis son introduction pour le jeu de Go, Monte Carlo Tree Search (MCTS) a été appliqué avec succès à d'autres jeux et a ouvert la voie à une famille de nouvelles méthodes comme Mutilple-MCTS ou Nested Monte Carlo. MCTS évalue un ensemble de situations de jeu à partir de milliers de fins de parties générées aléatoirement. À mesure que les simulations sont produites, le programme oriente dynamiquement sa recherche vers les coups les plus prometteurs. En particulier, MCTS a suscité l'intérêt de la communauté car elle obtient de remarquables performances sans avoir pour autant recours à de nombreuses connaissances expertes *a priori*.

Dans cette thèse, nous avons choisi d'aborder MCTS comme un système apprenant à part entière. Les simulations sont alors autant d'expériences vécues par le système et les résultats sont autant de renforcements. L'apprentissage du système résulte alors de la complexe interaction entre deux composantes : l'acquisition progressive de représentations et la mobilisation de celles-ci lors des futures simulations. Dans cette optique, nous proposons deux approches indépendantes agissant sur chacune de ces composantes. La première approche accumule des représentations complémentaires pour améliorer la vraisemblance des simulations. La deuxième approche concentre la recherche autour d'objectifs intermédiaires afin de renforcer la qualité des représentations acquises. Les méthodes proposées ont été appliquées aux jeu de Go et du Clobber solitaire impartial. La dynamique acquise par le système lors des expérimentations illustre la relation entre ces deux composantes-clés de l'apprentissage.

Mots-clés

Intelligence artificielle pour les jeux, Monte Carlo Tree Search, Apprentissage par renforcement, Jeux combinatoires, Computer-Go, Clobber Solitaire Impartial.

Title

Learning dynamics for Monte Carlo Tree Search : Application to combinatorial games.

Abstract

Monte Carlo Tree Search (MCTS) has been initially introduced for the game of Go but has now been applied successfully to other games and opens the way to a range of new methods such as Multiple-MCTS or Nested Monte Carlo. MCTS evaluates game states through thousands of random simulations. As the simulations are carried out, the program guides the search towards the most promising moves. MCTS achieves impressive results by this dynamic, without an extensive need for prior knowledge.

In this thesis, we choose to tackle MCTS as a full learning system. As a consequence, each random simulation turns into a simulated experience and its outcome corresponds to the resulting reinforcement observed . Following this perspective, the learning of the system results from the complex interaction of two processes : the incremental acquisition of new representations and their exploitation in the consecutive simulations. From this point of view, we propose two different approaches to enhance both processes. The first approach gathers complementary representations in order to enhance the relevance of the simulations. The second approach focuses the search on local sub-goals in order to improve the quality of the representations acquired. The methods presented in this work have been applied to the games of Go and Impartial Solitaire Clobber. The results obtained in our experiments highlight the significance of these processes in the learning dynamic and draw up new perspectives to enhance further learning systems such as MCTS.

Keywords

Artificial Intelligence on games, Monte Carlo Tree Search, Reinforcement learning, Combinatorial games, Computer-Go, Impartial Solitaire Clobber.

Remerciements

L'accomplissement d'une thèse de doctorat représente une épreuve sous bien des aspects avec quelques rares moments de reconnaissance et de longues périodes d'études... parfois difficiles, voire même de véritables traversées du désert. À cet égard, je tiens à remercier tout d'abord mes trois encadrants pour avoir proposé ce sujet mais aussi pour m'avoir accompagné pendant ces quatre années. Chacun d'eux s'est efforcé de m'orienter sur mon propre chemin par leur écoute et leur expertise respective.

Cependant aux difficultés scientifiques viennent s'ajouter les contingences de la vie qu'il faut malgré tout gérer au quotidien. C'est pour cette raison que mes remerciements vont en second lieu à l'État français pour m'avoir accordé pendant 3 années une bourse ministérielle me libérant des préoccupations matérielles de premier plan (ce n'est pas le cas de tous les doctorants ...). Ainsi, si je n'ai pas vécu d'eau fraîche pendant ces quatre années, j'ai pu pleinement mesurer l'amour de mes parents et plus généralement le soutien de mes amis.

C'est pourquoi je remercie tout particulièrement mon père et ma mère pour leur souci, parfois quotidien, à mon égard mais aussi mes deux frères (même s'ils ont leur propre façon de le montrer ...), ma Zia, mon Tonton ainsi que mes cousin(e)s. Le tableau familial ne serait pas complet si j'omettais de remercier chaleureusement les Gaëtan pour m'avoir recueilli lorsque ma fragile condition humaine s'est violemment rappelée à moi.

Sans qu'ils s'en rendent forcément compte, mes amis m'ont permis par leur simple présence de calmer mes petits nerfs éreintés par cette longue marche (ce qui est déjà pas mal !). Ainsi, je profite de cette tribune pour rendre hommage à mes grands amis de l'école primaire (ainsi qu'à leurs parents et amis respectifs), mes anciens collègues de l'INSA pour m'inciter à poursuivre mes recherches encore aujourd'hui, mes amis de la capoeira pour m'avoir forcé à débrancher mon cerveau pendant au moins 4 heures par semaine et mes collègues de labo pour m'avoir au contraire stimulé par nos échanges scientifiques, nos débats... mais aussi nos parties de cartes et autres jeux de société ; qui représentaient le prolongement évident d'un travail de thèse portant sur les jeux combinatoires (c'est tout du moins ce que je me disais alors pour avoir bonne conscience).

Un grand merci enfin aux relecteurs et relectrices (parfois acharnées) du présent document ainsi qu'à tous les lecteurs assidus dont font partie les membres du jury.

Table des matières

Introduction

Les jeux combinatoires couvrent des jeux pratiqués par les humains depuis des siècles comme les échecs ou bien le jeu de Go. À la différence de nombreux autres jeux pratiqués en société comme le Backgammon ou le Poker, l'issue de la partie dans un jeu combinatoire repose exclusivement sur les capacités intellectuelles des participants : les actions des joueurs ne dépendent pas d'un lancé de dés ou bien d'un tirage aléatoire. Ainsi, suivant cette perspective, les précurseurs en intelligence artificielle comme Turing ou Shannon ont souhaité confronter les capacités intellectuelles de leurs automates à celles d'un être humain [Rou14]. Encore aujourd'hui, les jeux combinatoires restent un champ d'application actif pour l'intelligence artificielle, étant donné la difficulté des problèmes traités.

La méthode Monte Carlo Tree Search a été conçue en particulier pour le jeu de Go. Contrairement aux précédentes méthodes, celle-ci évalue à la volée les différentes possibilités à partir de milliers de simulations aléatoires. Le retentissant succès qu'elle a obtenu pour ce jeu très difficile a suscité de nombreux travaux à son sujet ; en témoignent les thèses de doctorat successives à son sujet [Gel07 ; Sil09 ; Lew11 ; Shi11].

Dans ce travail en particulier, nous avons choisi de traiter cette méthode suivant une perspective inspirée des sciences cognitives. Ainsi nous considérerons ici les programmes reposant sur Monte Carlo Tree Search comme autant de systèmes cognitifs en perpétuel apprentissage. Cette démarche nous permet de mieux en cerner les limitations intrinsèques et de mettre en lumière certaines dynamiques internes à la méthode. Nous nous intéresserons plus spécifiquement à la dynamique d'apprentissage de cette méthode, à l'origine de son succès. Chacune des contributions présentées dans ce travail sera présentée au regard de cette dynamique d'apprentissage qui servira ainsi de fil conducteur.

Dans le chapitre 1, nous précisons la problématique abordée ainsi que les notions issues des sciences cognitives dont nous allons nous servir tout au long du document. Nous exposons notamment les limitations actuelles de Monte Carlo Tree Search du

point de vue des sciences cognitives et positionnons chacune de nos contributions à ce niveau de lecture.

Dans le chapitre 2, nous présentons les jeux sur lesquels nous avons travaillé ainsi que les axes suivant lesquels nous positionnons notre analyse. Pour l'essentiel, les travaux ont été réalisés avec le jeu de Go étant donné sa richesse mais nous avons aussi appliqué nos idées à un jeu plus simple à analyser : le jeu du Clobber Solitaire Impartial. Les axes d'analyse traitent de notions inspirées des sciences cognitives que nous avons approfondies au cours de ce travail.

Dans le chapitre 3, nous couvrons les représentations habituellement employées dans le champ de l'IA appliquée au jeu ainsi que leurs acquisitions. Les représentations traitées correspondent à l'ensemble des structures de données utilisées dans les programmes de jeu que ce soit pour la méthode de résolution ou bien en guise d'heuristique. Nous passons ensuite en revue les différentes méthodes d'apprentissage employées pour concevoir ou ajuster ces représentations en présentant leurs dynamiques d'apprentissage respectives.

Dans le chapitre 4, nous traitons de la méthode Monte Carlo Tree Search en particulier. Nous présentons son fonctionnement, sa dynamique d'apprentissage ainsi qu'un état de l'art des améliorations apportées à cette méthode dans le champ des jeux combinatoires. Nous profitons alors pour préciser dans quelle mesure nos contributions affectent la dynamique d'apprentissage par défaut de MCTS.

Dans le chapitre 5, nous présentons notre première contribution. Cette contribution propose de valoriser les représentations acquises à la volée par Monte Carlo Tree Search. La réutilisation de ces représentations sera l'occasion d'introduire une nouvelle dynamique d'apprentissage. Les résultats sont proposés pour le jeu Go exclusivement.

Dans le chapitre 6, nous présentons notre deuxième contribution. Il s'agit cette fois d'un modèle permettant d'ajuster à la volée la dynamique d'apprentissage de Monte Carlo Tree Search. Après avoir exposé le modèle général, nous évoquerons une mise en application de ce modèle pour les jeux de Go et du Clobber Solitaire Impartial.

1 Problématique

Les jeux combinatoires constituent un domaine d'application classique pour l'Intelligence Artificielle. Le champ de l'intelligence artificielle appliquée aux jeux (IAJ) a été un des témoins de l'avancée de la discipline comme par exemple la victoire de DeepBlue sur le champion mondial d'Échecs. Outre la dimension symbolique de la machine dépassant des joueurs humains, les motivations pour appliquer l'intelligence artificielle à des jeux sont multiples et toujours d'actualité. Dans la section 1.1, nous donnons un aperçu des principaux enjeux à appliquer l'IA à ce domaine, parmi lesquels celui que nous allons aborder.

Dans ce travail, nous avons choisi de traiter l'IA comme un système cognitif ; c'est à dire d'un système capable d'apprendre, de raisonner et d'interagir avec son environnement à l'image d'un être humain. Les sciences cognitives nous servent ici de cadre explicatif. Il ne s'agit pas d'un point de vue communément adopté au sein de la communauté de l'IAJ. Nous exposons sommairement les quelques notions que nous allons ensuite développer pour le domaine des jeux en section 1.2.

En particulier, nous considérons la méthode Monte Carlo Tree Search. Comme nous l'exposons dans la section 1.3, cette méthode d'IAJ est aussi bien une technique de résolution de jeu qu'un système cognitif limité. Nous évoquons enfin dans la section 1.4, le problème posé par cette méthode d'un point de vue cognitif puis nous présentons sommairement chacune de nos contributions et dans quelle mesure elles répondent à cette problématique.

1.1 Motivations et difficultés de l'IA appliquée aux jeux

Les IA ont été appliquées aux jeux combinatoires dès leurs débuts. Encore aujourd'hui, ils restent un champ d'application privilégié pour les techniques d'IA car il s'agit de problèmes simples à modéliser pour un ordinateur. Les raisons qui poussent à travailler sur les jeux combinatoires se sont progressivement précisées au fil des années. Dans la section 1.1.1 nous en présentons trois. Outre ces motivations, les jeux combinatoires sont intéressants car ils demeurent des problèmes très difficiles à résoudre pour des ordinateurs. Nous en présentons les deux principales raisons dans la section 1.1.2.

1.1.1 Motivations pour appliquer l'IA aux jeux

Étant donné sa nature interdisciplinaire, les motivations de l'IA recouvrent celles de multiples disciplines scientifiques. Dans une certaine mesure, les motivations pour appliquer l'IA aux jeux reflètent cette interdisciplinarité. Nous en présentons ici trois. La résolution théorique vise à établir des résultats mathématiques pour les jeux. La résolution pratique relève plus de l'informatique et la conception d'un système cognitif a trait aux sciences cognitives. Dans le cadre de ce travail, nous nous positionnerons sur la conception d'un système cognitif.

1.1.1.1. Résolution théorique

L'IA appliquée aux jeux a permis d'établir des résultats théoriques sur la résolution de certains jeux. La puissance de calcul des ordinateurs couplée aux méthodes adéquates permettent d'explorer intelligemment un grand nombre de possibilités, au point de « résoudre » le jeu en question [Sch+07]. La résolution d'un jeu recouvre plusieurs significations légèrement différentes quant à leurs implications pratiques. Dans le cadre de jeux à deux adversaires, Van Allis[All94] a défini trois niveaux de résolution cités ici par ordre croissant de force :

— Résolution très faible (*ultra-weakly solved* en anglais) : l'issue de la partie est connue si les deux joueurs jouent la meilleure stratégie possible. Cependant cette stratégie n'a pas encore été explicitée. C'est le cas par exemple pour le jeu de Hex. Il a été prouvé que le premier l'emporte toujours si les deux joueurs jouent parfaitement mais la stratégie optimale reste encore inconnue [1].

— Résolution faible (*weakly solved* en anglais) : l'issue de la partie est connue si les deux joueurs jouent la meilleure stratégie possible et, en plus, une stratégie optimale est connue depuis la disposition initiale. Face à un joueur sous-optimal, la stratégie cherchera systématiquement à atteindre l'issue identifiée au préalable. Cette stratégie ne sera pas capable de tirer profit des éventuelles erreurs commises par l'adversaire au cours de la partie. C'est le cas par exemple du jeu de Dames dont il a été prouvé que la partie se conclut en un match nul si les deux joueurs jouent optimalement, en détaillant une stratégie optimale depuis la position initiale [Sch+07].

— Résolution forte (*strongly solved* en anglais) : une stratégie optimale depuis n'importe quelle position est connue. Il n'est pas nécessaire que l'adversaire joue optimalement. Si jamais il commet une erreur, la stratégie optimale saura en tenir compte pour prendre l'avantage. C'est par exemple le cas pour le jeu

1. La preuve réalisée pour le jeu Hex n'est pas constructive : elle n'indique pas comment le joueur doit s'y prendre.

de Nim. Quel que soit l'état du jeu, il est possible de déterminer s'il s'agit d'une position gagnante ou perdante.

La résolution théorique d'un jeu est en principe l'apanage de la branche mathématique dédiée : la théorie des jeux combinatoires. L'IA appliquée aux jeux ne traite, à ce jour, que de la résolution faible. La résolution faible du jeu de Dames a ainsi été obtenue. Les méthodes alors déployées établissent par le calcul la séquence d'action optimale depuis la situation initiale. Ces preuves requièrent un lourd investissement en temps. Elles sont réalisées indépendamment de tout adversaire, en supposant que celui-ci agit de façon optimale. À la différence d'une résolution forte, ce type de résolution n'est pas en mesure de prodiguer l'action optimale à jouer en cours de partie.

1.1.1.2. Résolution pratique

La résolution pratique d'un jeu revient à concevoir un joueur artificiel capable d'emporter le plus souvent possible la victoire sur son adversaire. Il s'agit concrètement de pouvoir produire la meilleure action possible quel que soit la situation dans laquelle le programme se trouve. Suivant cette résolution, la qualité d'un joueur artificiel est jugé à l'aune de ses victoires. Les programmes doivent alors se plier aux contraintes des compétitions comme n'importe quel autre joueur comme, par exemple, décider de la prochaine action dans un temps limité. Des systèmes de classement comme le score Elo ou bien les Dan dans le jeu de Go permettent de suivre l'avancée des joueurs artificiels.

Dans une certaine mesure, la résolution pratique d'un jeu est l'approximation d'une résolution forte sans aucune garantie de performance. À titre d'exemple, dans des jeux comme les Dames ou les Échecs, les joueurs artificiels dépassent les performances des joueurs humains mais, à défaut de résultat théorique, il nous est encore impossible de savoir s'il existe une meilleure stratégie. La majorité des IA appliquées pour les jeux poursuivent cet objectif.

1.1.1.3. Conception d'un système cognitif

Les jeux combinatoires constituent un domaine d'application idéal pour la conception de systèmes cognitifs. Ils représentent en effet un problème dans lequel l'issue de la partie dépend exclusivement des capacités cognitives des participants. Par ailleurs, il s'agit d'un domaine où les compétences des joueurs artificiels sont facilement comparables à celles d'autres systèmes cognitifs complètement différents des joueurs humains. L'élaboration d'IA s'est inspirée de mécanismes cognitifs présents chez

les êtres humains, au travers notamment de la psychologie cognitive [Lan08]. À l'inverse, des techniques d'intelligences artificielles ont servi en retour à comprendre le fonctionnement des processus cognitifs des joueurs humains comme pour le jeu de Go par exemple [HBS12]. En suivant cette perspective, ce sont les facultés cognitives développées par le programme, plus que ses performances dans un contexte compétitif (avec un temps limité), qui nous intéresseront.

À défaut d'une autre source d'inspiration, les concepteurs d'IA se sont initialement inspirés du comportement de joueurs humains pour concevoir leurs programmes (il s'agissait d'ailleurs bien souvent de la même personne). L'introduction de mécanismes cognitifs dans les programmes actuels est toujours sujet à controverse. En effet le fonctionnement d'un ordinateur et d'un cerveau sont fondamentalement différents et il est légitime de relativiser un tel parallèle. Les défenseurs de ce point de vue comparent volontiers cette entreprise à celle des premières machines volantes inspirées d'oiseaux[2] : il est vain de mimer le battement d'ailes d'un oiseau pour réussir à voler. Nous pensons que l'analyse d'un programme informatique en tant que système cognitif permet de mieux comprendre son fonctionnement et d'ouvrir de nouvelles perspectives. Force est de constater que, pour certains jeux, les meilleurs joueurs mondiaux usent de leurs capacités cognitives et non d'un processeur. Par exemple dans le jeu de Go, les joueurs professionnels humains battent encore les meilleurs joueurs artificiels.

1.1.2 Problèmes posés à l'IA

Les jeux combinatoires habituellement traités par l'IA sont des jeux très complexes pour lesquels il n'existe pas de résolution théorique. La résolution pratique par l'intermédiaire d'un joueur artificiel se heurte à la forte combinatoire de ces jeux, c'est à dire au grand nombre d'évolutions possibles. Dans cette section nous évoquons les deux principales difficultés rencontrées par les méthodes d'IA. Tout d'abord la combinatoire des jeux traités est telle qu'elle empêche toute exploration exhaustive. D'autre part le programme ne repose que sur les connaissances qu'il possède pour juger de l'évolution de la partie au travers d'une fonction d'évaluation.

1.1.2.1. Résolution impraticable

Une approche naïve pour identifier le coup gagnant consiste à explorer exhaustivement des évolutions possibles jusqu'au bout et pour ensuite identifier l'action amenant à une issue gagnante quelle que soit la réponse de l'adversaire. En théorie,

2. Cette image a été utilisée par des membres de la mailing-list computer-go[Com09] pour illustrer l'opposition entre les programmes s'inspirant du comportement humain de ceux reposant sur la puissance de calcul de la machine.

une telle méthode donnerait ainsi l'action optimale quel que soit l'état considéré : elle serait alors équivalente à la résolution théorique forte. Toutefois le nombre d'état à considérer pour une telle méthode est bien trop grand pour pouvoir être mis en oeuvre. À titre d'exemple le nombre d'états total pour le jeu de Go en 19x19 est actuellement estimé à 10^{171}. En guise de comparatif, le nombre de particules élémentaires présentes dans l'univers observable est actuellement estimé [3] à 10^{85} . Une telle approche, à elle seule, est tout simplement impraticable (*untractable* en anglais) quelque soit la capacité de calcul à notre disposition.

1.1.2.2. Fonction d'évaluation

À défaut de pouvoir explorer toutes les évolutions possibles, le programme doit être capable de juger de lui-même chacune des situations atteintes ou bien envisagées à partir de la seule disposition des pièces. Dans les programmes d'IAJ, ce rôle est habituellement dévolu à la fonction d'évaluation qui pour chaque état calcule une valeur. La conception d'une fonction d'évaluation relève généralement de l'heuristique. Il est en effet nécessaire de correctement identifier, sans aucune garantie de performances, quelles sont les informations pertinentes et comment les traiter. Dans certains jeux comme les Échecs, la valeur habituellement attribuée à chacune des pièces permet d'établir une bonne approximation de l'avancement [4]. Cependant dans des jeux comme le Go, la conception *a priori* d'une telle fonction reste difficile[BC01].

1.2 Point de vue cognitiviste d'un programme de jeu

Dans le cadre de ce travail, nous abordons le programme de jeu comme un système cognitif à part entière, à l'instar de l'objectif présenté en section 1.1.1.3. Un programme de jeu est équivalent à un agent [5] en interaction avec un plateau de jeu. Il perçoit parfaitement l'état d'avancement de la partie (c'est à dire le plateau de jeu) ainsi que les actions de son adversaire. Il est aussi en mesure d'agir sur l'état du plateau par l'intermédiaire des actions qu'il joue (cf. Figure 1.1). En l'occurrence, l'agent joue de manière à remporter la partie sur son adversaire.

3. Il s'agit de la fourchette haute (source wikipedia `https://fr.wikipedia.org/wiki/Ordre_de_grandeur_(nombres)`

4. Notons cependant que les pièces possédées par chaque joueur ne suffisent pas à établir fidèlement l'avancement. La disposition des pièces est aussi déterminante : capturer une pièce maîtresse de l'adversaire, comme la reine, n'a en définitive pas beaucoup de valeur si la sienne se fait capturer au tour suivant.

5. Le terme agent est sujet à de nombreuses controverses en IA(ref what is an agent). Il est employé ici comme synonyme de programme ou de joueur artificiel afin de faciliter la correspondance avec d'autres branches de l'IA comme l'apprentissage par renforcement.

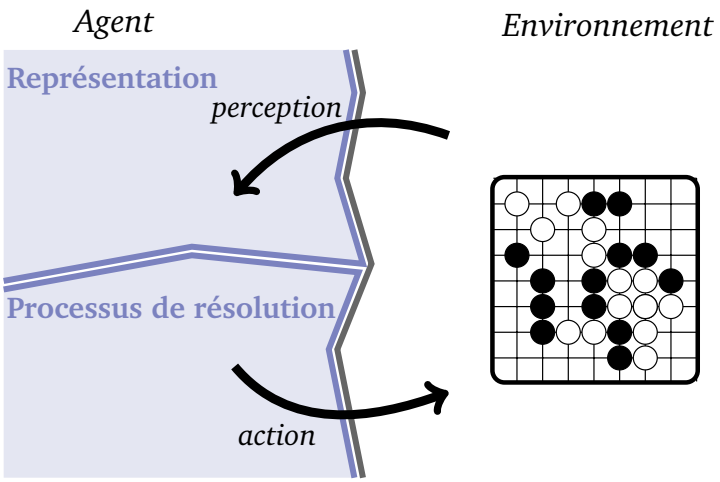


Figure 1.1.– Point de vue cognitiviste pour un programme de jeu

Les prises de décisions réalisées par l'agent tout au long de la partie représentent une tâche cognitive complexe. Nous distinguerons dans la mise en oeuvre de cette tâche deux composantes distinctes et internes à l'agent : les représentations et le processus de résolution. Les représentations correspondent aux données que possède l'agent sur le problème, et le processus de résolution indique le traitement à réaliser sur ces données pour atteindre l'objectif. Dans les sections qui suivent, nous détaillons un peu plus chacune de ces composantes en les illustrant par des exemples tirés de l'IA pour les jeux. Nous avons choisi volontairement des termes empruntés au langage des sciences cognitives par souci de commodité. En effet ces termes nous permettent de généraliser cette description à de nombreux programmes de jeux différents et ainsi faciliter la lecture du présent document. La notion de dynamique d'apprentissage sur laquelle nous allons ensuite revenir repose sur ces deux composantes internes de l'agent. Nous en profiterons aussi pour la définir sommairement dans une dernière section.

1.2.1 Représentation

En psychologie cognitive, le terme de représentation correspond aux symboles sur lesquels se baserait la pensée humaine[Lan08]. Par analogie, les représentations indiquent, pour un système artificiel, l'ensemble des informations sur lesquelles il va se baser pour élaborer un plan d'action. La définition de la représentation proposée couvre aussi bien la représentation du problème que la représentation du processus cognitif. Nous nous en tenons ici à une définition très large afin de pouvoir considérer l'ensemble des heuristiques ou structures de données employées dans les programmes de jeu. Dans le chapitre 3 nous précisons plus avant les caractéristiques de ces représentations.

Définition 1.1: Représentation

Une représentation correspond à toute information ou donnée utilisée par l'agent pour atteindre son objectif. Ainsi une représentation couvre aussi bien la perception de l'état du plateau de jeu que des tactiques de jeu. Une représentation peut intégrer d'autres représentations plus simples par l'intermédiaire de structures de données. Les représentations sont généralement évaluées afin de pouvoir plus facilement les comparer.

Dans le cadre de l'IA appliquée aux jeux, les représentations couvrent l'ensemble des connaissances expertes habituellement utilisées en guise d'heuristiques mais aussi des structures de données plus complexes comme les arbres de recherche ou bien les réseaux de neurones artificiels. Les fonctions qui associent une valeur à chaque état de l'environnement sont aussi considérées comme des représentations à l'image des fonctions d'évaluation des méthodes comme $\alpha - \beta$ ou bien des fonctions de valeur d'action Q^π en apprentissage par renforcement.

1.2.2 Processus de résolution

À chaque tour, l'agent doit décider d'une nouvelle action à jouer. La prise de décision requiert de nombreux traitements. Nous distinguerons ici les traitements produisant de nouvelles représentations de ceux aboutissant au choix d'une action. Le processus de résolution correspond ici au traitement qui décide d'une action à jouer.

Définition 1.2: Processus de résolution

Un processus de résolution est un traitement réalisé sur les représentations de l'agent aboutissant à la mise en application d'une action. Cette action peut être celle jouée effectivement par l'agent ou bien toute mise en application simulée en interne. Un processus de résolution est toujours motivé par un objectif à atteindre qui correspond, en principe, à celui poursuivi par l'agent.

Bien évidemment, la méthode dans son ensemble constitue un processus de résolution à part entière. Toutefois au cours de sa réflexion, l'agent sera généralement amené à se mettre en situation de décider de la prochaine action. Ces décisions internes seront alors autant de processus de résolution plus simples. C'est le cas par exemple dans de nombreuses méthodes de l'IAJ classique. Le parcours d'un arbre de recherche suivant l'élagage $\alpha - \beta$ correspond à plusieurs processus de résolution successifs. En particulier dans les méthodes d'apprentissage par renforcement, le processus de résolution est partie prenante de l'apprentissage de l'agent. Il s'agit

de la politique d'action π justement qui décide de la prochaine action au cours de l'apprentissage.

1.2.3 Dynamique d'apprentissage

Lorsque l'agent apprend, les représentations sont progressivement ajustées à partir d'autres représentations fournies à l'agent ou bien à partir d'un processus de résolution. La description des différentes composantes contribuant à cette évolution ainsi que leurs interactions constituent ce que nous appellerons la dynamique d'apprentissage.

Définition 1.3: Dynamique d'apprentissage

Dans le cadre d'un système apprenant, la dynamique d'apprentissage restitue les interactions entre les représentations et les processus de résolution participant à l'apprentissage. Il est possible de représenter la dynamique d'apprentissage comme un graphe où chaque noeud correspond à une composante (représentation ou un processus) et un arc témoigne d'une interactions entre les deux composantes.

Si la dynamique d'apprentissage indique l'existence d'une interaction, elle n'en précise pas la teneur exacte. Il s'agit ici d'une illustration schématique du déroulement de l'apprentissage.

Les représentations apprises automatiquement sont généralement élaborées à partir d'autres représentations comme par exemple dans l'apprentissage supervisé où l'agent apprend par l'observation de parties déjà jouées. Cependant d'autres méthodes d'apprentissage comme l'apprentissage par renforcement proposent d'apprendre à partir de parties générées par le processus de résolution de l'agent ; créant ainsi une interaction entre la représentation apprise et le processus de résolution. Nous reviendrons plus longuement sur la dynamique d'apprentissage de ce type d'apprentissage dans le chapitre 3.

1.3 Points de vue pour Monte Carlo Tree Search

Au cours de ce travail, nous nous intéressons à la méthode Monte Carlo Tree Search. Cette méthode a été initialement conçue pour les jeux combinatoires et s'est très vite imposée comme une référence. En outre, cette méthode a la particularité de pouvoir être traitée suivant deux perspectives différentes : comme une méthode de résolution pratique ou bien comme un système cognitif. Dans cette section, nous présentons la méthode à l'aune de chacune de ces perspectives. Ces points de vue

font naturellement écho aux raisons avancées en section 1.1.1. Le second point de vue, nous servira de trame de fond tout au long de ce manuscrit.

1.3.1 MCTS comme un programme informatique

MCTS est avant tout une méthode pour la résolution pratique de jeux. À titre d'exemple, tous les meilleurs joueurs artificiels de Go implémentent MCTS à ce jour. Cette méthode décide de la prochaine action à jouer à partir du résultat de milliers de simulations de fin de parties. À la différence des autres méthodes de résolution classiques de l'IAJ, la fonction d'évaluation est établie à la volée à partir du seul résultat des simulations. Cette méthode se prête particulièrement bien aux jeux pour lesquels il est difficile d'établir une fonction d'évaluation comme le Go [Gel+12] ou le General Game Playing[MC08].

Si les méthodes de Monte Carlo ont été appliquées pour le Go depuis plus de 20 ans [Brü93], elles ont depuis bénéficié de l'essor des puissances de calcul des machines ainsi que de l'effort constant de chercheurs du domaine[BH04 ; Bou05]. MCTS fait à présent partie des méthodes de résolution classique de l'IA, à l'instar de méthodes comme $\alpha - \beta$.

1.3.2 MCTS comme un système cognitif

Suivant la perspective adoptée dans ce travail, MCTS définit aussi un système cognitif. En effet l'agent apprend à évaluer les états à partir de milliers de simulations qui sont autant d'expériences simulées en interne. Les connaissances ainsi apprises sont spécifiques à la situation traitée et le système en apprend de nouvelles à chaque nouveau tour de jeu. Néanmoins MCTS met en oeuvre une véritable dynamique d'apprentissage au fur et mesure des simulations sur laquelle nous allons régulièrement revenir tout au long de ce travail.

Les méthodes de Monte Carlo indiquent n'importe quelle méthode ayant recours à des simulations aléatoires. Dans le cadre de processus de décision séquentiels comme les jeux combinatoires, les méthodes de Monte Carlo comme MCTS sont souvent présentées comme des méthodes d'apprentissage par renforcement[SB98]. Cette théorie nous servira de support pour étayer notre propos.

1.4 Présentation des contributions pour MCTS

MCTS a considérablement amélioré les performances des joueurs artificiels pour des jeux comme le Go [GS08 ; Gel+12]. Cependant les meilleurs programmes ne rivalisent pas encore avec les meilleurs joueurs mondiaux pour ce jeu, malgré le recours à d'importantes puissances de calcul. Dans la première section, nous en esquissons la raison suivant une perspective cognitiviste. Nous présentons ensuite les deux contributions proposées en regard de la limitation avancée.

1.4.1 Difficultés posées à MCTS

Les performances de MCTS dépendent du nombre de simulations que l'agent est capable de réaliser au cours d'un tour. À cet effet, de nombreuses améliorations ont porté sur l'accélération du générateur de simulation [6] ou bien sur la multiplication du nombre de simulations en parallélisant la méthode sur plusieurs machines[Bou+10 ; Seg11]. Cependant, malgré un accroissement conséquent du nombre de simulations, les performances du programme stagnent progressivement suivant une courbe logarithmique[Bou+10] : tout gain supplémentaire de performances nécessite alors un ajout de simulations exponentiellement plus grand.

En poursuivant la perspective cognitiviste, l'agent apprend alors à évaluer de plus en plus de représentations différentes mais n'en tire pas suffisamment d'enseignements. La quantité de représentations apprises est privilégiée au détriment de la qualité. La nature des représentations ainsi que la dynamique d'apprentissage restent les mêmes tout au long de l'apprentissage. En l'état actuel, l'agent n'est pas en mesure de capitaliser de cette volumineuse expérience des connaissances lui servant en retour à améliorer ses futures expériences simulées ou bien concentrer sa recherche sur certaines zones particulières du plateau. D'ailleurs, les principales avancées ont été obtenues par l'ajout de connaissances supplémentaires fournies *a priori* au programme [GS08]. Notre perspective vise à terme à ce que le programme soit capable d'apprendre ces connaissances supplémentaires, et même d'autres, de lui-même.

1.4.2 Première contribution : valoriser les représentations apprises

MCTS apprend par défaut des représentations spécifiques à des situations très précises. Les connaissances apprises par l'agent ne se prêtent pas à être réutilisées dans d'autres situations. La première contribution présentée dans le cadre de ce travail, propose d'extraire des représentations apprises d'autres représentations transver-

6. Au sein de la communauté, la librairie LibEgo a souvent été mise en avant pour la rapidité avec laquelle les simulations aléatoires sont réalisées [Lew11]

sales et de les réutiliser dans d'autres contextes. En particulier, nous avons choisi ici de valoriser ces connaissances dans les simulations aléatoires afin d'accentuer l'apprentissage de l'agent.

1.4.3 Deuxième contribution : adapter la dynamique d'apprentissage

Tout au long de la partie, MCTS accumule les représentations suivant la même dynamique. L'agent n'est pas en mesure d'adapter son apprentissage suivant la situation. La deuxième contribution présentée dans le cadre de ce travail, permet à l'agent d'adapter à la volée sa dynamique d'apprentissage à l'instar des représentations initialement apprises. Chacune de ces dynamiques permettra notamment à l'agent de mieux orienter sa recherche.

1.5 Synthèse

Dans ce chapitre nous avons tout d'abord identifié trois motivations principales pour mettre en œuvre une IA pour les jeux :

— la résolution théorique se rapproche d'une vision mathématicienne et cherche à établir un résultat absolu pour le jeu ;

— la résolution pratique se rapproche d'une vision informaticienne et cherche à obtenir les meilleurs résultats possibles relativement à son adversaire dans un contexte compétitif (temps limité) ;

— la conception d'un système cognitif s'approche d'une vision cognitiviste et cherche à améliorer les capacités d'apprentissage et de réflexion du système.

Dans une perspective cognitiviste, un programme de jeu est identifiable à un agent indépendant doué de perceptions et d'actions. Lors d'une prise de décision, il est possible de distinguer deux composantes différentes :

— les représentations qui correspondent à toutes les informations sur lesquelles repose la décision de l'agent ;

— le processus de résolution qui traite les représentations pour décider de l'action à jouer.

Ces deux composantes interagissent, lors de l'apprentissage de l'agent, au travers d'une dynamique d'apprentissage que nous allons progressivement préciser au cours de ce manuscrit.

Nous traitons ici spécifiquement de la méthode MCTS. Cette méthode représente à la fois une redoutable méthode pour la résolution pratique de certains jeux, mais aussi un agent en perpétuel apprentissage grâce à une expérience simulée.

Cependant, malgré un accroissement des puissances de calcul, le méthode n'arrive pas à venir à bout des difficultés inhérentes à tous les jeux combinatoires. D'après un point de vue cognitif, l'agent n'est actuellement pas en mesure de pouvoir tirer suffisamment profit de cette expérience virtuelle pour améliorer son apprentissage. Les deux contributions présentées dans ce travail visent à améliorer les capacités d'apprentissage de l'agent : la première en valorisant les représentations acquises, la deuxième en adaptant la dynamique d'apprentissage de l'agent.

2 Positionnement

L'IAJ traite de nombreux jeux qu'ils soient populaires comme les Échecs ou les Dames ou bien d'inspiration mathématique comme Hex ou Clobber. Chacun de ces jeux dispose de propriétés différentes qui rendent leur traitement plus ou moins difficile par un joueur artificiel. Dans la section 2.1, nous motivons le choix des jeux que nous allons considérer dans ce travail. Ces jeux ont été retenus avant tout pour la difficulté qu'ils posaient à des joueurs artificiels, dans leur fonctionnement actuel, mais aussi parce qu'ils étaient propices au développement de leurs facultés cognitives.

Suivant cette perspective, nous approfondissons ensuite, dans la section 2.2, les deux notions inspirées des sciences cognitives que nous souhaitons mettre en avant dans ce travail. Ces deux éclairages nous serviront aussi bien pour décrire l'état de l'art que pour présenter chacune de nos contributions. Les notions abordées dans ce chapitre seront ensuite progressivement précisées pour l'IAJ et la méthode Monte Carlo Tree Search au cours des chapitres suivants.

2.1 Jeux considérés

Au cours de cette thèse, nous allons nous concentrer sur deux jeux pour mener nos expérimentations : les jeux de Go et du Clobber Solitaire Impartial. Le jeu de Go a été choisi pour les difficultés qu'il pose à tout système cognitif. Toutefois l'analyse du comportement d'un joueur artificiel peut s'avérer complexe. C'est pourquoi nous nous sommes aussi intéressé au jeu du Solitaire Impartial (CSI) pour essayer nos modèles sur un environnement mieux maîtrisé mais néanmoins difficile pour IA. Les motivations pour chacun de ces jeux sont respectivement approfondies dans les sections 2.1.1 et 2.1.2.

2.1.1 Motivations pour le jeu de Go

Le jeu de Go est un jeu combinatoire séculaire originaire d'Asie. Au fil des générations de joueurs qui se sont succédées, il s'est développé toute une pratique et une culture autour de ce jeu populaire : pratiques pédagogiques, livres de techniques. Dans le champ de l'IAJ, la communauté du computer-go est elle aussi active. Après la défaite de Kasparov face à DeepBlue, le jeu de Go représente l'un des derniers

jeux combinatoires populaires pour lequel les meilleurs joueurs humains dépassent encore les machines.

Le jeu de Go représente la « nouvelle drosophile » de l'IAJ. Ce jeu illustre à merveille les difficultés posées par les jeux combinatoires aux IA, que nous avons présentées en section 1.1.2. Tout d'abord sa résolution est particulièrement impraticable. À cet égard, le jeu de Go est bien plus difficile comparativement aux autres jeux combinatoires populaires comme nous le verrons en section 2.1.1.1. De surcroît, l'élaboration d'évaluation interne pour le jeu de Go est là encore difficile à réaliser. Nous en expliquerons les raisons dans la section 2.1.1.2. Au demeurant, les meilleurs joueurs professionnels sont encore au-dessus des intelligences artificielles. Une partie de cette énorme difficulté est naturellement levée par les joueurs humains grâce à leur capacité cognitive. Ainsi le jeu de Go constituerait davantage un problème cognitif qu'un problème calculatoire comme nous l'évoquons en section 2.1.1.3. Pour toutes ces raisons, et cette dernière en particulier, le jeu de Go constitue l'application par excellence pour concevoir nos contributions. Les règles du jeu sont présentées ci-dessous dans le paragraphe du même nom.

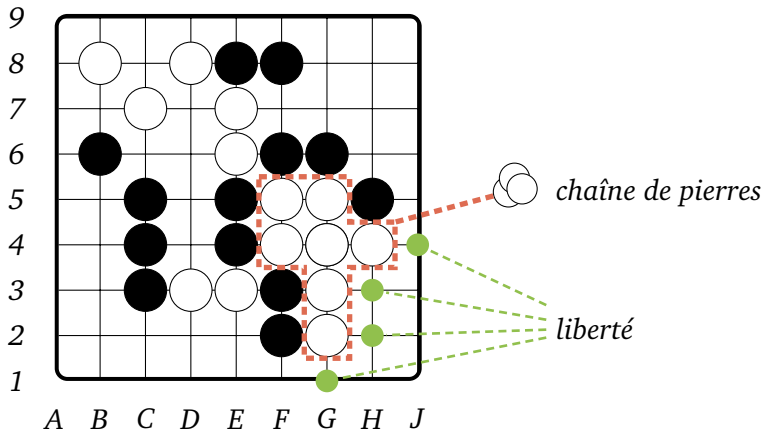


Figure 2.1.– Illustration des notions de chaîne de pierre et deliberté dans le jeu de Go.

Règles du jeu Le jeu de Go se joue à 2 : un joueur blanc et un joueur noir. La taille standard du plateau, appelé goban, est de 19x19 mais des parties rapides ou d'initiation peuvent se jouer en 9x9. Le goban démarre vide, puis chaque joueur pose à tour de rôle une pierre de sa couleur. Une fois posées, les pièces ne bougent plus jusqu'à ce qu'elles se fassent capturer. Une pierre, ou une chaîne de pierres connexes, est capturée dès qu'elle ne possède plus d'intersections vides autour d'elle, appelées libertés (cf. figure 2.1). Les configurations locales des pierres sont de fait déterminantes dans ce jeu. Le joueur qui l'emporte à la fin de la partie est celui détenant le plus grand territoire. Pour plus d'informations sur le calcul des scores finaux ou les règles détaillées du jeu, merci de vous référer au site `sensei.xmp.net`.

2.1.1.1. Forte combinatoire

Malgré son ancienne origine, il compte parmi les jeux combinatoires les plus difficiles à ce jour. Il appartient à l'une des classes de problèmes les plus difficiles suivant la théorie de la complexité algorithmique : savoir si une position quelconque est gagnante est un problème EXPTIME-COMPLETE. En pratique, la résolution du jeu de Go est particulièrement impraticable (au sens présenté dans la section 1.1.2.1). En effet le nombre d'états atteignables dépasse de loin celui d'autres jeux combinatoires comme cela est présenté dans la table 2.1. En outre, le facteur de branchement [1] élevé rend l'exploration de l'espace de recherche d'autant plus difficile.

Jeu	Nombre d'états estimés	Facteur de branchement
Jeu de dames anglaises	10^{18}	2.8
Jeu d'Échecs	10^{47}	35
Jeu de Go 19x19	10^{171}	250

Table 2.1.– Récapitulatif du nombre d'états possibles et du facteur de branchement des jeux combinatoires classiques[All94]

2.1.1.2. Évaluation interne difficile

S'il est relativement simple pour un joueur professionnel d'évaluer une position, les chercheurs du domaine ont d'énormes difficultés à l'encoder dans une fonction d'évaluation (cf. section 1.1.2.2). Contrairement à des jeux comme les échecs, les pierres posées sur le goban n'ont aucune valeur en soi, ce sont les relations qu'elles entretiennent les unes aux autres qui sont déterminantes. Par ailleurs, la taille du plateau est relativement grande et il est difficile d'identifier les informations importantes dans une position.

2.1.1.3. Problème cognitif

La principale force des intelligences artificielles réside dans leur puissance de calcul contrairement aux joueurs humains qui ne sont pas capables de réaliser de longs calculs mais disposent de plus grandes facultés cognitives. Les joueurs humains ont recours à de multiples abstractions de la disposition des pierres pour identifier, dans une position, les informations importantes. Cette capacité d'abstraction, assez innée chez les humains, fait encore défaut aux intelligences artificielles. Ainsi une

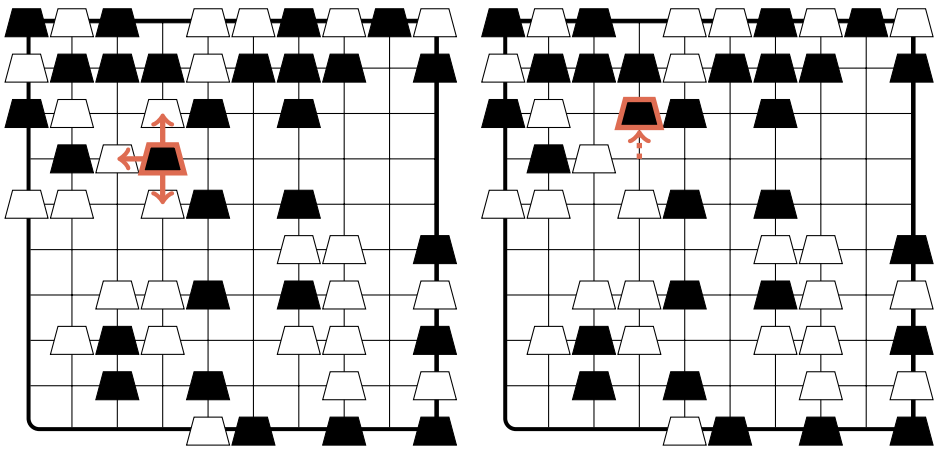
1. Le facteur de branchement correspond au nombre moyen d'actions à chaque tour.

perspective pour les joueurs artificiels consiste à essayer d'étendre leurs capacités cognitives plus que leur capacité de calcul.

2.1.2 Motivations pour le jeu du Clobber Solitaire Impartial

Le jeu du Clobber a été inventé par Albert, Grossmann et Nowakowski en 2001 à Games-at-Dal. Il a été introduit auprès de la communauté lors du séminaire organisé à Dagstuhl en février 2002 [Dem+02]. Depuis ce séminaire, le jeu de Clobber a fait l'objet de compétitions entre programmes informatiques lors des Computer Olympiad à l'instar du jeu de Go. Toutefois sa relative jeunesse ne lui confère pas la même notoriété que ce dernier. Le jeu de Clobber se joue initialement à 2 joueurs mais des variantes à 1 joueur, dites solitaires, ont été inventées depuis sa création. Nous traitons ici les règles pour la version Solitaire et Impartiale du Clobber[BDG13].

Le jeu du Clobber Solitaire Impartial a été choisi pour sa simplicité d'analyse : il nous permet en effet d'étudier le comportement de notre programme sur un environnement un peu plus maîtrisé que celui du jeu de Go. La section 2.1.2.1 précise les raisons d'une telle maîtrise. Néanmoins ce jeu pose aux intelligences artificielles, des difficultés communes à celles d'autres jeux combinatoires. En particulier le jeu de Clobber dispose d'un fort facteur de branchement lors des premiers tours comme nous le présentons en section 2.1.2.2. En outre la prise de décision pour ce jeu est rendue encore plus difficile par la présence d'actions critiques comme nous l'évoquons en section 2.1.2.3. Pour toutes ces raisons, le jeu du Clobber Solitaire Impartial constitue un intéressant banc d'essai pour essayer nos méthodes. Les règles du jeu sont présentées ci-dessous dans le paragraphe du même nom.



(a) Action possibles **(b)**Action jouée

Figure 2.2.– Illustration d'une action dans le jeu de Clobber Solitaire Impartial.

Règles du jeu Le jeu du Clobber Solitaire Impartial se joue qu'à un seul joueur. En principe, il est jouable sur n'importe quel graphe. Nous avons choisi ici une

grille de taille $n \times m$, à l'image d'un goban. La disposition classique des pierres en début de partie alterne une pierre noire et une pierre blanche à chaque intersection (comme les cases d'un damier). À chaque tour le joueur choisi indifféremment une pierre blanche ou noire et « assomme [2] » une pierre adjacente (horizontalement ou verticalement) de la couleur opposée. La pierre qui a été assommée est retirée du plateau et la pierre « assommant » prend sa place comme présenté en figure 2.2. Comme tous les jeux solitaires, l'objectif du joueur est de prolonger le plus longtemps possible sa partie.

2.1.2.1. Environnement maîtrisé

À la différence du jeu de Go, le Clobber Solitaire Impartial se joue à un seul joueur. Le programme bénéficie alors d'une plus grande maîtrise de son environnement : il sait exactement quelle sera sa future position pour chaque action. Le résultat obtenu par le programme dépendra exclusivement des choix qu'il aura réalisés auparavant : l'issue du jeu ne sera pas influencée par les actions d'un adversaire opposé à ses propres plans. Un jeu solitaire est, en définitive, un problème de planification dans un environnement déterministe et à information parfaite. Il s'agit d'identifier la bonne séquence d'action à réaliser ; c'est à dire de définir le meilleur ordre pour jouer ces actions dans un environnement relativement simple. En particulier, les résultats démontrés indiquent le meilleur score atteignable, à savoir la taille de la plus grande séquence d'actions possibles. Les performances atteintes par les joueurs artificiels sont confrontables à celles établies théoriquement (cf. Annexe A.1).

2.1.2.2. Forte combinatoire

La difficulté d'un jeu est étroitement liée au nombre d'actions possibles à chaque tour. La prise de décision est généralement rendue plus difficile lorsque le nombre d'actions disponibles à chaque tour est élevé car le joueur doit savoir mieux discerner les bonnes des mauvaises actions.

Suivant la théorie de la complexité, le jeu du Clobber Solitaire Impartial appartient à une catégorie de problèmes bien plus simples que le du jeu de Go : identifier le score maximum atteignable depuis une configuration de pierres quelconque et NP-COMPLET (sur une grille). Cependant le facteur branchement (cf. note 1) est très élevé en début de partie particulièrement. À titre d'exemple, le nombre de positions accessibles à une séquence de 3 actions de la position initiale, est d'au moins 10 millions environ pour un plateau de 8x8 avec les pierres disposées en damier. En guise de comparaison, le nombre de positions accessibles à une profondeur de 3

2. *to clobber* en anglais signifie « assommer »

est égal à 511 920 dans une partie de Go à deux joueurs sur un plateau 9x9 et 46 millions dans une partie sur un plateau 19x19 [3]. À mesure que la partie avance, le facteur de branchement pour le jeu Solitaire décroît conjointement avec le nombre de pierres restantes. Néanmoins le nombre de positions accessibles en début de partie, donne une idée de la forte combinatoire de ce jeu, pour une taille de plateau relativement petite.

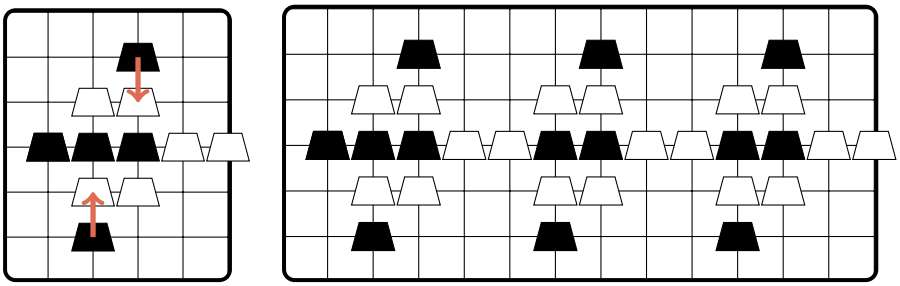
2.1.2.3. Décisions difficiles

Pour le Clobber Solitaire, l'importance de l'ordre suivant lequel sont jouées les actions est très variable. Dans de nombreuses situations, l'ordre avec lequel sont jouées deux actions est parfaitement permutable. Par exemple, dès lors qu'il existe deux composantes de pierres disjointes, l'ordre entre des actions jouées dans chacune de ces composantes est parfaitement interchangeable. À l'inverse d'autres actions sont critiques car elles ont des conséquences irréversibles sur le cours de la partie. Le moment au cours duquel sont jouées ces actions est crucial. Afin de prolonger au maximum la partie, il convient de maintenir le plus longtemps possible les pierres connexes. Les actions particulièrement critiques sont celles susceptibles de séparer un groupe de pierres connexes en deux composantes disjointes. À partir du moment où cette action sera jouée, les composantes évolueront indépendamment. La figure 2.3 illustre cette notion d'action critique au travers d'une configuration spécialement conçue à cet effet.

2.2 Notions des sciences cognitives approfondies

Comme nous l'avons évoqué précédemment, nous souhaitons aborder un joueur artificiel comme un système cognitif à part entière. Suivant cette démarche, nous allons approfondir tout au long de ce manuscrit certaines notions issues de ce domaine dans le cadre d'un programme de jeu. Ces notions sont implicitement présentes dans de nombreux travaux de l'état de l'art. Cette grille de lecture nous permettra de les couvrir suivant une perspective plus large et de souligner l'apport de chacune de nos contributions. En particulier, nous traiterons des notions d'abstraction de la représentation et de rétroaction dans les section 2.2.1 et 2.2.2. L'abstraction de la représentation qualifie la façon dont sont formalisées des connaissances complexes de l'environnement tandis que la rétroaction correspond aux informations perçues par l'agent qui sont, notamment, à la base de toute dynamique d'apprentissage.

3. Les chiffres avancées ne tiennent pas compte des invariances par rotation, translation voire des positions identiques.



(a) Schéma critique **(b)**Schéma répété

Figure 2.3.– Situation critique pour le jeu de Clobber Solitaire Impartial : La situation de la figure 2.3a se réduit à une seule pierre avec le bon choix d'action. Depuis cette situation il y a exactement 14 actions possibles (deux à chaque fois qu'une pierre noire est adjacente à une pierre blanche). Cependant seules les 2 actions indiquées en rouge permettent d'atteindre l'objectif d'une seule pierre à la fin : soit un ratio de 1 action sur 7. C'est à dire 14% de chances de sélectionner l'une de ces actions suivant un tirage aléatoire uniforme. Toutes les autres actions laissent nécessairement une pierre en plus. Il est possible de faire gonfler le facteur de branchement en mettant bout à bout plusieurs fois cette même configuration comme dans la figure 2.3b. Le ratio restera cependant constant.

2.2.1 Abstraction dans la représentation

Dans un agent, les représentations correspondent à l'ensemble des connaissances que l'agent mobilise pour prendre sa décision (cf. Définition 1.1 p. 9). Les représentations vont de la simple disposition des pièces sur un plateau à des considérations stratégiques pour les tours à venir. Parmi toutes ces représentations différentes, les représentations abstraites correspondent à celles proposant une description plus synthétique d'un état du jeu en s'intéressant à certaines zones spécifiques du plateau ou bien en introduisant des concepts supplémentaires à ceux présents dans les règles du jeu. L'élaboration de représentations abstraites reste encore à ce jour une pierre d'achoppement des intelligences artificielles car ces représentations introduisent généralement de nombreux biais non désirés. Comme nous l'évoquons dans la section 2.2.1.1, il est néanmoins indispensable d'y recourir pour les jeux combinatoires. Nous présentons ensuite deux manières différentes de mettre en oeuvre cette abstraction dans les représentations :

— Abstraction par restriction ;

— Abstraction par interprétation ou montée en abstraction.

Nous reviendrons dans les sections suivantes sur chacune de ces abstractions en les illustrant avec des exemples tirés des jeux. Ces deux formes d'abstractions ne sont pas exclusives et sont parfois combinées dans la pratique.

2.2.1.1. Nécessité d'abstraire la représentation

La représentation la plus élémentaire et n'introduisant aucune ambiguïté consiste à traiter chaque état du jeu indépendamment. Cependant, comme nous l'avons évoqué dans la section 1.1.2.1, cette représentation ne peut pas suffire à elle seule pour décider de la prochaine action, tant le nombre de représentations à traiter serait grand pour les jeux combinatoires.

L'abstraction de la représentation permet de réduire considérablement le nombre de représentations à considérer pour l'agent. D'une part, il s'agit de représentations s'appliquant à de nombreux états différents. D'autre part, une fois ces représentations acquises, l'agent n'a pas à les apprendre de nouveau car elles sont justement réutilisables. Néanmoins ces abstractions introduisent généralement des biais qu'il est plus ou moins difficile de déceler selon les situations.

2.2.1.2. Abstraction par restriction

L'abstraction par restriction ne s'intéresse qu'à une sous-partie de l'état d'avancement du jeu. Dans la description complète d'un état, toutes les informations ne seront pas nécessairement utiles pour décider de la prochaine action. L'abstraction par restriction cherche alors à concentrer l'attention sur les zones les plus importantes. Dans le cadre d'un jeu comme le Go, cette forme d'abstraction est souvent employée pour traiter de connaissances tactiques. Par exemple les formes réalisées localement par les pierres servent à identifier des actions généralement meilleures que d'autres. De même, les dernières actions jouées donnent une bonne indication des zones contestées pour la situation courante.

La difficulté pour cette forme d'abstraction est de correctement identifier quelles sont les informations pertinentes. En oubliant une certaine partie des informations présentes dans l'état du jeu, cette abstraction introduit un biais plus ou moins important suivant les cas de figure. S'il existe des règles générales pour toutes les situations, comme les motifs formés par les pierres au Go, certaines informations ne deviennent décisives qu'au regard de la situation courante.

2.2.1.3. Abstraction par interprétation

L'abstraction par interprétation indique lorsque de nouveaux concepts, non présents initialement dans les règles, sont introduits pour simplifier la représentation. Des parties de la représentation peuvent être identifiées à ces concepts et ainsi amener la réflexion à un niveau plus synthétique de discussion. Si ces termes n'ont pas un

ancrage précis dans l'état d'avancement du jeu ils permettent d'en faciliter l'analyse. Dans le cadre du jeu de Go, de nombreux concepts sont ainsi introduits par les joueurs humains pour faciliter la description de situations de jeu. Des concepts comme les notions de groupe, de vie et de mort de ces groupes, voire d'attaque et défense d'un groupe ne sont pas perceptibles d'eux-mêmes. Toutes ces notions nécessitent une interprétation, par l'agent, de la position des pierres.

La difficulté pour cette forme d'abstraction est d'identifier et de définir les concepts introduits. Dans le cadre du jeu de Go, la pratique du jeu par des joueurs humains a permis de dégager de nombreux concepts que les précédentes IA pour le jeu (avant l'introduction de MCTS) cherchèrent à transcrire [BC01]. Cependant de tels concepts ne se transposent pas facilement à un ordinateur et restent ensuite difficiles à exploiter. Par ailleurs ces concepts sont fournis *a priori* à l'agent qui n'est pas en mesure de les adapter suivant la situation.

2.2.2 Qualité et mécanisme d'exploitation de la rétroaction

Dans le cadre d'un apprentissage hors-ligne ou bien au cours d'une partie, l'agent considère de nombreuses observations différentes. Parmi ces observations, la rétroaction désigne celles qui amènent l'agent à modifier ses composantes internes : c'est à dire ses représentations ou bien son processus de résolution. La rétroaction permet notamment d'ajuster progressivement le comportement d'un agent au cours du temps à l'image d'un missile à tête chercheuse ajustant progressivement sa trajectoire. Dans la section 2.2.2.1, nous verrons dans quelle mesure une rétroaction est nécessaire pour un joueur artificiel. La pertinence des connaissances ajustées seront dépendantes aussi bien de la qualité de la rétroaction que de son interprétation. Dans les sections 2.2.2.2 et 2.2.2.3 respectivement, nous présentons chacun de ces deux aspects en les agrémentant d'exemples.

2.2.2.1. Nécessité d'une rétroaction

Dans le cadre d'un long processus de décisions séquentielles comme une partie de Go, il est impossible de prévoir à l'avance quelle sera son évolution exacte. L'agent doit en permanence adapter son comportement aux actions réalisées par son adversaire. Il peut s'agir des actions effectivement jouées par l'adversaire ou bien de celles planifiées en interne. Par exemple, la fonction d'évaluation est habituellement employée comme dans la méthode $\alpha - \beta$ afin d'orienter correctement la recherche suivant les actions envisagées.

Dans le cadre d'un agent apprenant, la définition d'une rétroaction permet d'automatiser le processus d'apprentissage. Ces informations serviront à ajuster pro-

gressivement les représentations et seront le point de départ de toute dynamique d'apprentissage.

2.2.2.2. Qualité de la rétroaction

La rétroaction constitue l'information de référence pour ajuster le comportement de l'agent. L'origine de cette information et sa précision auront une influence décisive sur la qualité des connaissances acquises. Par exemple, lorsque l'agent a lui-même participé à la production de cette rétroaction, il disposera de plus d'informations quant à sa fiabilité. À l'inverse, si la rétroaction provient d'une entité indépendante, l'agent ne sera pas en mesure de juger *a priori* de sa fiabilité. Indépendemment de leur origine, certaines rétroactions ne comportent aucune ambiguïté, comme le résultat d'un match entre deux joueurs ou bien la disposition des pièces en fin de partie. Des rétroactions aussi précises servent notamment à établir des constatations empiriques fiables. D'autres rétroactions sont plus heuristiques, en revanche, et introduisent de fait un biais avant même d'interpréter l'information.

Dans les méthodes de Monte Carlo, la rétroaction systématiquement employée est le résultat de la simulation aléatoire, qui est en principe sans ambiguïté. De plus l'agent sera plus à même de juger de la qualité des évaluations produites. D'une façon plus générale lorsque la fonction d'évaluation est conçue hors-ligne, la qualité de la rétroaction est intrinsèque à son concepteur ou à la méthode d'apprentissage employée pour la concevoir.

2.2.2.3. Mécanismes d'exploitation de la rétroaction

Les mécanismes d'exploitation de la rétroaction couvrent aussi bien le traitement de cette information, que les composantes de l'agent qui en bénéficieront. Le traitement de la rétroaction précise les modifications apportées à l'information lors de son traitement pour tenir compte du contexte suivant lequel la rétroaction a été perçue comme par exemple le délai entre une prise de décision et la perception de la rétroaction. Les composantes correspondent généralement aux représentations ajustées. À titre d'exemple, les rétroactions perçues par l'agent ne sanctionnent pas systématiquement l'action venant d'être réalisée par celui-ci. C'est à l'agent d'interpréter parmi les actions qu'il a réalisées jusqu'à présent, celles qui ont le plus contribué à un tel résultat et dans quelle mesure [4].

4. Ce problème porte plus généralement le nom *Temporal Credit Assignment* dans la littérature[SB98] et concerne en particulier les problèmes de décisions séquentielles.

Le choix des représentations modifiées déterminera la dynamique d'apprentissage de l'agent (cf. Définition 1.3 à la page p. 10). Tout d'abord les seules représentations qui seront apprises ou ajustées sont celles bénéficiant d'une rétroaction. Les autres resteront statiques tout au long du processus. Ensuite, un choix précautionneux des représentations renforcées permet de créer des phénomènes de bouclage auto-catalytique dans la dynamique d'apprentissage. En effet, lorsque la représentation renforcée interagit avec le processus de résolution à l'origine de la rétroaction, le processus sera entraîné en retour pour accentuer les prochaines rétroactions, à l'image de l'effet de Larsen dans les systèmes de sonorisation. Dans ce travail, nous verrons à plusieurs reprises ce phénomène à l'œuvre. Il s'identifie naturellement dans la dynamique d'apprentissage par une interaction à double sens entre une représentation et un processus de résolution.

2.3 Synthèse

Dans ce chapitre, nous avons tout d'abord motivé les deux jeux sur lesquels nous allons mener nos expérimentations :

— Le jeu de Go a été choisi pour l'extrême difficulté qu'il pose aux joueurs artificiels mais aussi parce qu'il est particulièrement propice à l'élaboration de capacités cognitives, comme en témoignent celles de joueurs professionnels. Les modèles proposés dans les chapitre 5 et 6 seront appliqués à ce jeu ;

— Le jeu du Clobber Solitaire Impartial pour la difficulté naturelle qu'il pose à des joueurs artificiels mais aussi la grande maîtrise dont nous disposons grâce aux résultats théoriques. Il s'agit notamment d'un environnement où il est plus facile d'analyser le comportement de l'agent. Seul le modèle présenté dans le chapitre 6 sera appliqué à ce jeu.

Nous avons ensuite présenté les notions inspirées des sciences cognitives que nous avons souhaité aborder au cours de ce travail :

— L'abstraction de la représentation traite de la définition des représentations manipulées par l'agent. La difficulté des jeux combinatoires est telle, qu'il est indispensable de se restreindre à un nombre limité de représentations. Les choix opérés pour formaliser ces différentes abstractions introduiront nécessairement un biais lors de la résolution. Nous préciserons cette notion dans le chapitre 3.

— Le choix et le traitement de la rétroaction traitent de l'ajustement dynamique des représentations manipulées par l'agent. Cette notion est étroitement liée à la dynamique d'apprentissage au travers notamment des phénomènes de bouclage auto-catalytique. Nous reviendrons sur cette notion en particulier dans le chapitre 3 pour les méthodes d'apprentissage et dans le chapitre 4 pour la méthode MCTS en particulier.

3 Représentations et apprentissage de représentations

La question de la représentation fait partie des développements fondamentaux de l'intelligence artificielle. L'IA appliquée aux jeux a suivi les différentes évolutions du domaine à ce sujet depuis ses débuts. Ces évolutions portent aussi bien sur les informations véhiculées par ces représentations que sur la manière de les acquérir. Les programmes actuels mobilisent de nombreuses représentations d'origines et de portées différentes. Chacune de ces représentations revêt une importance spécifique dans le processus de décision. En outre, différentes représentations interagissent souvent ensemble afin d'améliorer les compétences du joueur rendant, de fait, leur analyse plus complète.

Dans ce chapitre nous détaillons tout d'abord de manière plus approfondie les différentes représentations. Nous présentons les différentes contributions dans le domaine des jeux dans la section 3.1. Les différentes connaissances associées seront décrites suivant un panel de propriétés que nous présentons au préalable. Ensuite nous abordons l'origine de ces représentations dans la section 3.2. Nous traitons notamment dans cette partie des fondements théoriques de MCTS qui est issue de l'apprentissage par renforcement. C'est à cette occasion que nous introduisons plus en détail le concept de dynamique d'apprentissage. Enfin un résumé en section 3.3 récapitule les principaux points abordés au cours de ce chapitre.

3.1 Connaissances dans le domaine des jeux

Conformément à la définition proposée dans le chapitre 1, les représentations couvrent ici un large panel allant de la perception d'un état à la structure de données complexes en passant par des heuristiques spécifiques. C'est pourquoi, nous proposons dans un premier temps plusieurs propriétés pour classifier les différentes représentations en section 3.1.1. Ensuite nous recensons les principales représentations employées dans le domaine des jeux en section 3.1.2. Chacune de ces connaissances sera détaillée suivant ses propriétés et ses limitations intrinsèques. Un tableau récapitulatif en section 3.1.3 résume cet état de l'art au sujet des représentations.

3.1.1 Propriétés

Les propriétés rendent compte des choix opérés pour concevoir des connaissances appréhendables par une machine. En effet, à défaut de savoir précisément comment concevoir une représentation abstraite, il existe plusieurs façons de généraliser des représentations à plusieurs états. Les propriétés rendent compte de différentes généralisations possibles. Chacune de ces propriétés a des conséquences sur l'usage et le stockage de ces connaissances. Nous en avons identifié trois :

— La dimension de la représentation ;

— Le traitement de la représentation ;

— La couverture de la représentation.

La dimension relève plus d'un trait descriptif d'une représentation suivant qu'elle couvre la dimension spatiale ou temporelle du problème. Les deux propriétés suivantes renvoient aux deux formes d'abstraction identifiées dans le chapitre 2. Le traitement de la représentation correspond à l'abstraction par interprétation ou montée en abstraction tandis que la couverture de la représentation correspond à l'abstraction par restriction. La première nécessite l'introduction de notions supplémentaires aux spécifications du problème tandis que la seconde ne s'intéresse qu'à une partie des observations.

3.1.1.1. Dimension de la représentation

La dimension de la représentation précise la dimension de l'environnement décrite par ces connaissances, au sens physique du terme. Dans le cadre des jeux combinatoires, nous distinguons par exemple deux dimensions différentes : temporelle et spatiale. Les deux dimensions ne sont pas exclusives et il est possible de concevoir des connaissances spatio-temporelles. La dimension d'une représentation a des conséquences quant à sa généricité ou la concision des informations qu'elle porte.

Définition 3.1: Représentation temporelle

Une représentation temporelle décrit le déroulement temporel des actions. La structure des données en mémoire renvoie à l'ordre suivant lequel sont survenus les éléments. Elle restitue mieux la dynamique d'une partie.

Les représentations temporelles décrivent le déroulement d'un processus de décision séquentiel : ce qui les rend en principe génériques à n'importe quel jeu. Par ailleurs les informations fournies par des représentations temporelles expriment mieux la causalité entre les actions : elles permettent de mettre en avant les dépendances entre des actions consécutives. Par exemple au Go, la dernière action jouée par

l'adversaire attend parfois une réponse immédiate à proximité au risque de perdre des pans de territoires.

Définition 3.2: Représentation spatiale

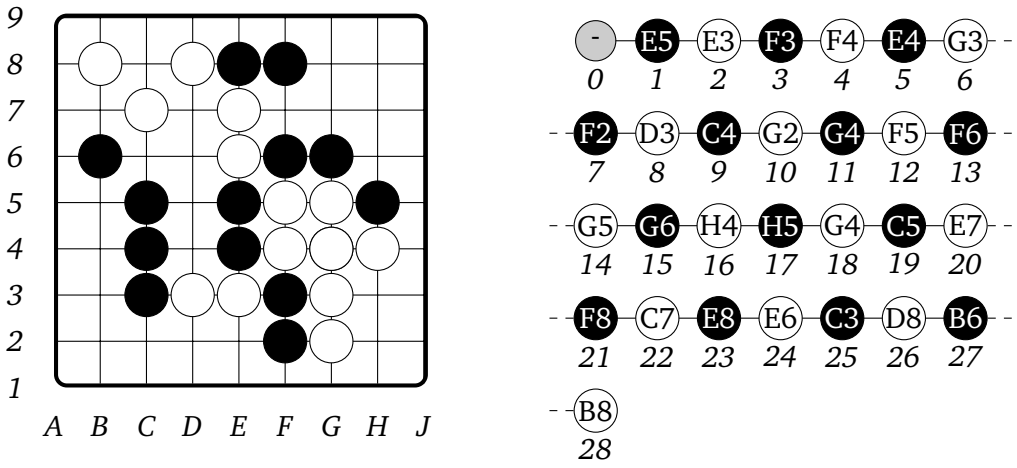
Une représentation spatiale décrit la disposition spatiale de pièces sur le plateau. La structure des données en mémoire renvoie à une position sur un plateau. Elle restitue mieux l'état de la position.

Les informations fournies par des représentations spatiales sont en général plus synthétiques que des représentations temporelles pour décrire un état : elles fournissent un maximum de renseignements sur une position à partir de très peu de données. C'est le type de représentations utilisées par exemple pour proposer des casse-têtes dans des jeux comme les échecs ou le Go. Cependant elles sont souvent spécifiques au jeu concerné car les pièces et leur placement varient d'un plateau à l'autre.

Exemple illustratif La figure 3.1 représente une position de Go suivant les deux dimensions présentées : spatiale et temporelle. Toutes les deux décrivent exactement le même état. La représentation spatiale restitue l'état actuel du jeu (la disposition actuelle des pièces sur le plateau) tandis que la représentation temporelle rappelle la séquence d'actions réalisées depuis le début de la partie. Cependant ces deux représentations ne sont pas équivalentes. Il est ainsi plus simple de suggérer la prochaine action à jouer depuis la représentation spatiale. À l'inverse, la représentation temporelle restitue mieux le déroulement étape par étape de cette partie opposant le programme Fuego au joueur professionnel Hsun Chou en particulier[Mül10]. En effet, pour une même représentation spatiale d'une position il existe plusieurs représentations temporelles différentes possibles. Les deux représentations temporelles correspondent alors à deux transpositions différentes du même état de jeu.

3.1.1.2. Le traitement d'une représentation

Le traitement d'une représentation indique l'écart entre la description absolue de l'état de l'environnement et la représentation relative à l'agent pour ce même état. Nous distinguons les représentations brutes constituées d'informations élémentaires, de celles constituées d'informations plus ou moins enrichies par des traitements successifs. Le traitement donne une indication sur la quantité de connaissances différentes que devra apprendre l'agent.



(a) Plateau de jeu (spatial) **(b)***Séquence d'action (temporel)*

Figure 3.1.– Comparaison entre une représentation spatiale et une représentation temporelle : Les deux représentations décrivent exactement la position de Go suivant deux dimensions différentes. La représentation temporelle proposée est une séquence d'actions possibles aboutissant à cet état. Nous pouvons noter que toutes les actions jouées dans la séquence d'action n'apparaissent pas nécessairement dans la représentation spatiale résultante. Suivant la représentation temporelle l'intersection G4 a tout d'abord été noire au tour 11 pour ensuite devenir blanche au tour 18.

Définition 3.3: Représentation brute

Une représentation brute est constituée exclusivement des informations susceptibles d'être perçues dans l'environnement (position des pièces, actions des joueurs), sans aucun traitement supplémentaire.

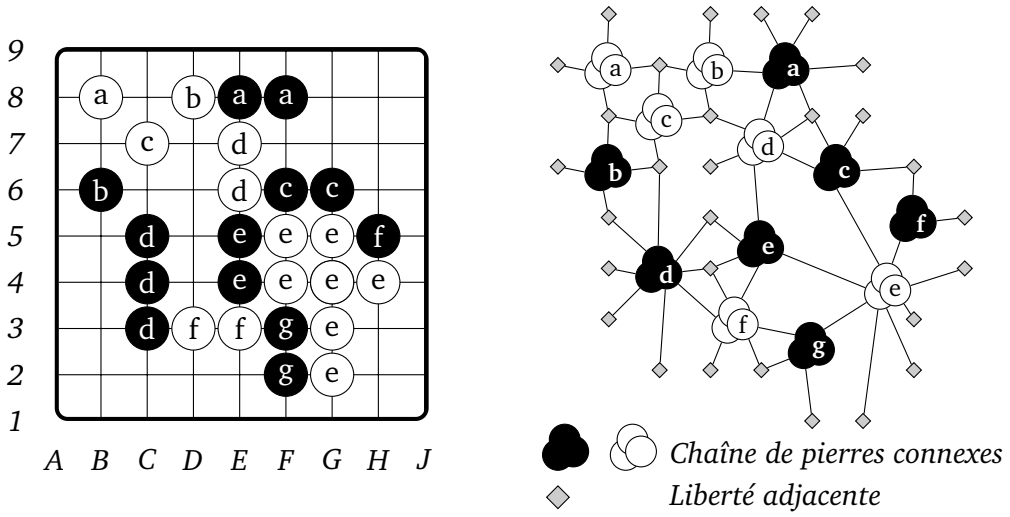
Les informations de ces représentations ne sont complétées d'aucune interprétation supplémentaire et sont en principe plus génériques que des représentations enrichies. L'interprétation de ces informations est alors laissée au processus de l'agent ou à d'autres représentations les intégrant. Néanmoins les représentations brutes sont tellement sensibles à des détails de l'environnement que l'agent doit en général disposer de nombreuses représentations brutes pour pouvoir efficacement répondre à un problème. D'ailleurs, le nombre de représentations brutes possibles dépend de la taille de l'espace des états.

Définition 3.4: Représentation enrichie

Une représentation enrichie est constituée en partie d'informations calculées par l'agent à partir d'autres représentations (élémentaires ou non).

Les traitements réalisés sur les informations servent en général à concentrer l'attention sur les éléments les plus importants d'une observation et limite ainsi la taille des informations perçues. Ces traitements sont généralement spécifiques au jeu pour lequel ils ont été conçus et rendent de telles représentations moins génériques à d'autres jeux. En outre le traitement des données brutes est une première forme d'interprétation de ces informations. Cette interprétation introduit un biais lors de la mise en application de ces connaissances qu'il est difficile de contrôler.

Exemple illustratif La figure 3.2 représente la même position de Go suivant que les représentations soient brutes ou bien enrichies. La représentation spatiale des pièces correspond à une description brute de l'environnement : chacune des pièces est représentée. Le *Common Fate Graph* est une représentation enrichie proposée initialement par Graepel, Goutrie, Krüger et Herbrich [Gra+01] : cette représentation n'est pas fournie explicitement à l'agent qui doit la calculer en interne. Ce traitement repose sur le concept de liberté qui est fondamental au jeu de Go[1] . Une telle représentation est plus synthétique de l'état de l'environnement et rend mieux compte des enjeux tactiques d'une situation pour le jeu de Go.



(a) Plateau de jeu (brute)

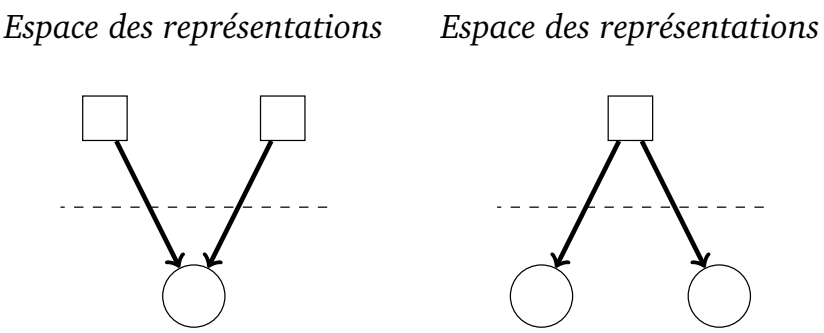
(b) *Common Fate Graph (enrichie)*

Figure 3.2.– Comparaison entre une représentation brute et une représentation enrichie pour une position de Go : Les deux représentations décrivent exactement le même état pour deux traitements différents. Les lettres sur les pierres sont indiquées afin de faciliter la correspondance entre les deux représentations.

1. La liberté d'une pierre ou d'un ensemble contiguë de pierres est le nombre d'intersections vides qui sont adjacentes. Au Go, une pierre complètement entourée de pierres adverses (c'est à dire avec 0 libertés) est capturée.

3.1.1.3. La couverture d'une représentation

La couverture d'une représentation renseigne sur la portée applicative des représentations. Nous distinguons les représentations précises qui rendent compte d'une description exhaustive de l'état de l'environnement, de celles dont l'application est plus floue, qui ne considère qu'une partie de l'état. La couverture rend compte de la diversité des situations qu'un agent sera susceptible de traiter à partir de ces représentations. La figure 3.3 est une représentation schématique des différences entre les deux couvertures.



(a) Précise (*many-to-one*) **(b)**Flou (*one-to-many*)

Figure 3.3.– Représentations précises et floues : la couverture d'une représentation est semblable aux notions d'injectivité et de surjectivité habituellement employées pour décrire une application (au sens mathématique du terme). Les espaces de départ et d'arrivée correspondent respectivement à l'espace des représentations et l'espace des états. La couverture indique dans quelle mesure une représentation couvre l'espace des états.

Définition 3.5: Représentation précise

Une représentation précise ne couvre qu'un seul et unique état de l'environnement. Les informations qu'elle véhicule, concernent un seul état. Il est toutefois possible que plusieurs représentations précises distinctes traitent du même état (cf. figure 3.3a).

Des représentations précises distinguent chacun des états possibles de l'espace de recherche. Il n'existe alors aucune ambiguïté quant à leur interprétation. Une telle précision garantit notamment la convergence du comportement de l'agent vers la solution optimale dans des méthodes d'apprentissage comme l'apprentissage par renforcement (cf. section suivante). Cependant ces représentations ne sont applicables qu'à un seul état au maximum. Pour un jeu combinatoire, une représentation précise implique en général que cette représentation soit aussi brute : à défaut d'une

plus grande connaissance sur le jeu. Le nombre de représentations nécessaires pour atteindre l'objectif fixé est susceptible d'être important [2].

Définition 3.6: Représentation floue

Une représentation floue couvre plus d'un état de l'environnement. Les informations qu'elle véhicule, sont applicables à plusieurs état différents (cf. figure 3.3b).

Les représentations floues sont réutilisables pour différentes situations permettant d'en limiter le nombre nécessaire, à l'image des représentations enrichies. Cependant cette application multiple induit souvent des ambiguïtés quant à l'interprétation des informations portées par ces connaissances. En effet ces informations sont susceptibles d'être parfois appliquées à des états auxquels elles ne correspondent pas du tout. Plus une représentation est floue, plus elle s'applique à de nombreux états et plus il est difficile de s'assurer qu'elle ne sera pas employée à tort.

Exemple illustratif La figure 3.4 pose en vis-à-vis une représentation précise d'un état de l'environnement avec plusieurs représentations floues. À nouveau la représentation spatiale des pièces est une représentation précise de l'état de l'environnement car il s'agit de la simple transcription de chacune de ses variables. Les informations associées à cette représentation ne concernent que cet état précis de l'environnement. Les schémas spatiaux décrivent quant à eux des portions du plateau (les schémas spatiaux seront détaillés dans la section suivante). Les informations qu'ils véhiculent sont applicables à de nombreux états possibles. Nous présentons ici des schémas spatiaux qu'il est possible d'appliquer à cette situation comme à d'autres.

3.1.2 État de l'art

Nous allons couvrir les différentes représentations employées pour les jeux combinatoires. Un inventaire exhaustif dépasse le cadre de ce travail tant le domaine a été prolifique en la matière. Nous nous sommes concentrés ici autour de celles habituellement employées dans les programmes de Go parmi lesquels nous comptons :

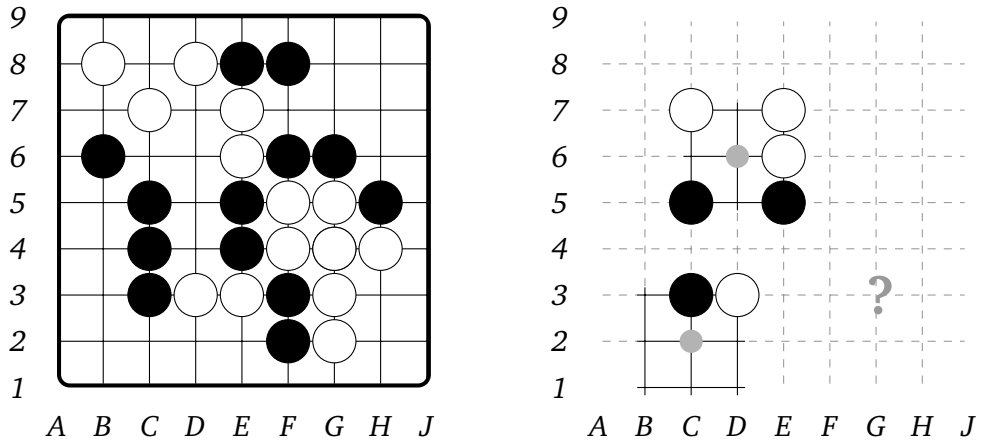
— Les arbres de recherches ;

— Les schémas partiels (spatiaux, temporels et experts) ;

— Les réseaux de Bradley-Terry ;

— Les réseaux de neurones.

2. Dans d'autres domaines, des représentations floues suffisent à répondre à l'objectif initialement fixé. C'est par l'exemple l'un des enjeux des codes identifiants en théorie des graphes[KCL98].



(a) Plateau de jeu (précise) **(b)**Schémas spatiaux (floue)

Figure 3.4.– Comparaison entre une représentation précise et floue pour une position de Go : Les représentations présentées sont valides pour le même état de l'environnement. La description du plateau de jeu précise chacune des intersection tandis que les différents schémas spatiaux ne spécifient qu'un sous-ensemble des intersections. Par conséquent les schémas spatiaux sont valides pour cette situation comme cela est présenté mais pourraient s'appliquer à d'autres états. Les lettres et chiffres dans la figure de droite sont indiqués afin de faciliter la correspondance entre les deux représentations.

Nous les présentons ici au regard des propriétés précédemment définies. Ces propriétés caractérisent leur usage dans un programme de jeu et induisent certaines des limitations identifiées. Ces limitations se révéleront notamment lorsqu'elles seront parties prenantes de méthodes plus complexes comme Monte Carlo Tree Search. Nous y reviendrons plus précisément sur ces connaissances lors du chapitre 4.

3.1.2.1. Arbre de recherche

Les arbres de recherche formalisent une exploration systématique des différentes issues possibles dans un problème. Il s'agit de l'approche la plus immédiate pour résoudre un problème combinatoire. Déjà en 1887, Édouard Lucas décrit la solution d'un problème combinatoire sous la forme d'un arbre de recherche dans ses récréations mathématiques [Luc87]. Les arbres de recherche sont employés naturellement en IA dans des problèmes de planification par exemple [RN10].Ils permettent ainsi de peser chacune des différentes options offertes à l'agent dans des problèmes décisionnels. Dans le cadre de l'IA appliquée aux jeux, des méthodes classiques comme $\alpha - \beta$ ou MCTS reposent notamment sur la construction d'arbres de recherche.

Définition 3.7: Arbre de recherche

Un arbre de recherche est une représentation brute, précise et temporelle, qui structure efficacement les différents états atteignables pour des séquences d'actions croissantes. Le noeud à la racine de l'arbre correspond à l'état courant et chaque noeud fils indique l'action à appliquer à l'état du jeu atteint au niveau du noeud père. La profondeur d'un noeud indique la taille de la séquence d'actions consécutives le séparant de l'état courant. Ainsi les noeuds immédiatement reliés à la racine correspondent aux états à une seule action de l'état courant. Les noeuds reliés à ceux-ci sont à deux actions consécutives de l'état courant et ainsi de suite (cf. figure 3.5).

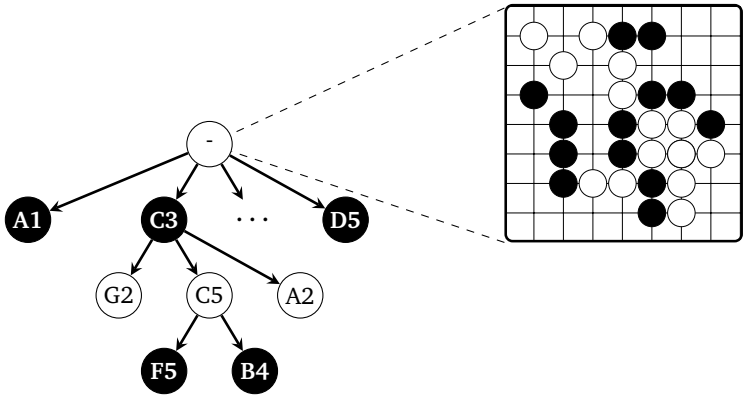


Figure 3.5.– Illustration d'un arbre de recherche : le noeud à la racine correspond à l'état courant (schématisé par le plateau) et chaque noeud fils à une évolution de cet état.

L'information contenue dans les arbres est essentiellement portée par les noeuds. En effet chacune des évolutions couvertes par un noeud de l'arbre est évaluée permettant ainsi de mieux comparer les différentes options. Tel que nous l'abordons, cette représentation est essentiellement temporelle mais en pratique elle intègre aussi une représentation spatiale à chaque noeud sous la forme d'une table de transposition de manière à identifier deux séquences d'actions différentes qui aboutissent à un même état [3].

Limitation - effet d'horizon Les états couverts par un arbre de recherche correspondent à ceux immédiatement proches de l'état courant par construction. Un arbre juge des états susceptibles d'être atteints dans les prochains tours. Cependant un arbre de recherche ne perçoit qu'un nombre limité d'états (proches de l'état actuel) car une exploration exhaustive de telles représentations brutes est impraticable. Au delà des états couverts, l'arbre ne fournit plus aucune indication. En pratique cela s'illustre par un manque d'anticipation plus généralement appelé effet d'horizon.

3. Un arbre de recherche avec table de transposition n'est plus un arbre au sens strict de la théorie des graphes mais un graphe orienté sans cycle.

Définition 3.8: Effet d'horizon

L'effet d'horizon est le phénomène suivant lequel une exploration plus approfondie de l'arbre change radicalement l'évaluation des états ; comme par exemple une défaite imminente dans les tours à venir. Cette limitation est intrinsèque aux arbres de recherche car la profondeur de la recherche est toujours limitée au premières actions immédiatement consécutives de l'état courant.

Des joueurs professionnels ont parfois exploité implicitement cette faiblesse afin de battre leurs adversaires artificiels. Par exemple dans une partie opposant le programme Fuego au joueur professionnel Chun Hsun Chou, ce dernier a volontairement posé un piège visible 10 tours plus tard. Le programme n'a pas su l'anticiper et le joueur professionnel a remporté la partie alors qu'il se trouvait auparavant dans une situation perdante selon lui[Mül10].

Limitation - redondance d'information Un arbre de recherche considère pour chacun de ses noeuds des représentations précises et brutes. Par conséquent deux séquences d'actions qui diffèrent d'une seule action sont rigoureusement différentes et ne partagent aucune information. Dans de nombreuses situations pourtant, cette différence n'aura aucune influence sur l'évaluation. Ce manque d'abstraction dans les représentations impliquent que des informations semblables se retrouvent dans différentes branches de l'arbre de recherche [Dra09]. Nous parlerons de redondance d'information.

Définition 3.9: Redondance d'information

La redondance d'information indique dans un arbre de recherche qu'une information similaire est susceptible de se retrouver dans différentes branches de l'arbre de recherche. Par exemple, une même sous-séquence d'actions résolvant un problème local devra être évaluée à nouveau dès que les actions précédant cette sous-séquence diffèrent, c'est à dire pour chaque branche de l'arbre. Cette limitation est intrinsèque des représentations brutes et précises dans les noeuds de l'arbre de recherche.

3.1.2.2. Schémas partiels

Les schémas partiels formalisent des prédicats élémentaires sous la forme d'une réponse associée à un contexte particulier (cause et conséquence). Le contexte propose une représentation partielle de l'environnement pour lequel le schéma est valable. La réponse indique l'action à produire par l'agent pour ce contexte. Les schémas

couvrent un large panel d'heuristiques employées dans les jeux combinatoires. Nous distinguerons ici trois catégories de schémas suivant les propriétés de leur contexte

— les schémas temporels ;

— les schémas spatiaux ;

— les schémas enrichis ou experts.

Définition 3.10: Schéma temporel

Un schéma temporel est un schéma partiel dont le contexte correspond à la séquence d'actions ayant immédiatement précédé l'état courant. La réponse indique l'action à jouer ensuite. Il s'agit de connaissances brutes, floues et temporelles (cf. figure 3.6).

Plus formellement, un schéma temporel de taille n est une suite ordonnée de n actions consécutives. L'action en position n est la réponse et les $n-1$ premières actions sont le contexte. Par extension, toute séquence d'actions peut être perçue comme un schéma temporel.

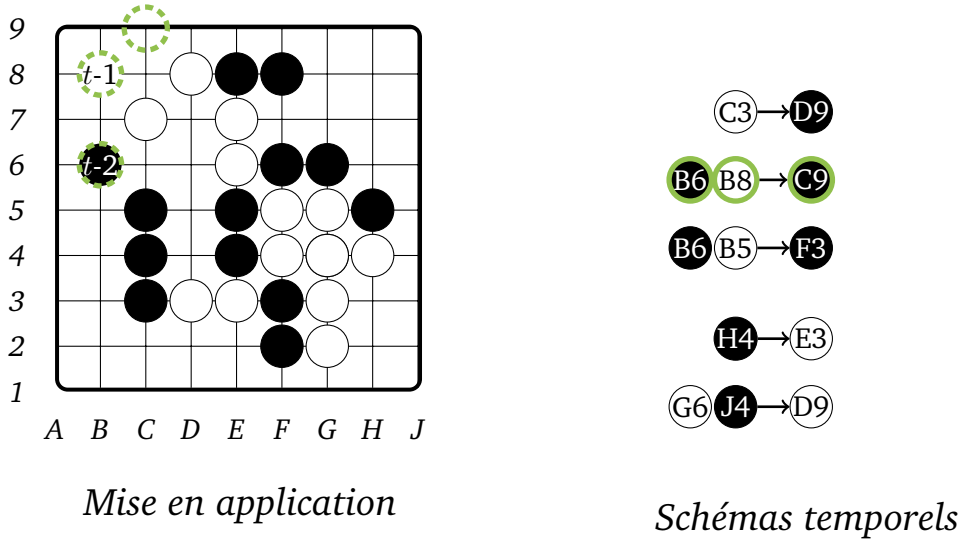


Figure 3.6.– Exemple de schéma temporel : parmi les schémas temporels présentés seulement le deuxième en partant du haut est applicable à la situation de jeu proposé. Il suggère en l'occurrence de répondre aux actions précédemment jouées par le coup C9.

La dimension temporelle est identique pour tous les problèmes de décision séquentiels à la différence de la dimension spatiale. Ces schémas sont en principe génériques à n'importe quel jeu. Il sont notamment employés dans le domaine du General Game Playing : des méthodes dites de N-gram ou LGRF reposent sur des schémas temporels. Ce type de schéma partiel se prête bien à modéliser la dynamique d'une partie comme, par exemple apporter des réponses spontanées à une agression de

l'adversaire. La taille du contexte est généralement limitée à une ou deux actions consécutives.

Définition 3.11: Schéma spatial

Un schéma spatial est un schéma partiel dont le contexte correspond à une portion contiguë du plateau appelé motif (*pattern* en anglais). La réponse indique l'action à jouer ensuite (elle est généralement située au centre du motif). Il s'agit de connaissances brutes, floues et spatiales (cf. figure 3.7).

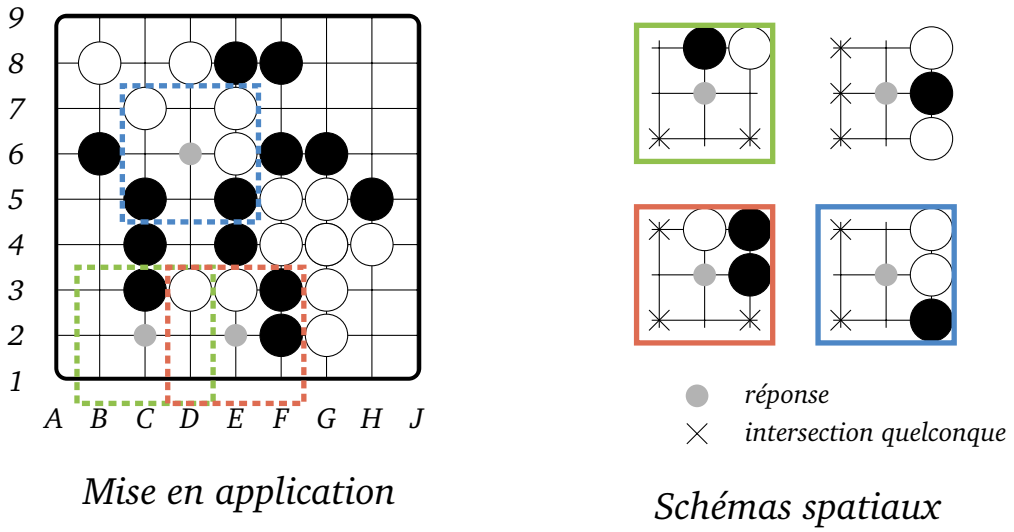


Figure 3.7.– Exemple de schéma spatial : parmi les schémas spatiaux présentés, seuls les trois schémas encadrés sont applicables à la situation de jeu présentée. Chaque schéma suggère de jouer respectivement au centre de la zone correspondante.

Les motifs spatiaux sont les plus employés pour le jeu de Go. En effet, le jeu de Go est très visuel et les joueurs raisonnent souvent à partir des formes décrites par les pierres sur le plateau. Ces schémas servent à intégrer sous une forme visuelle des rudiments de tactiques lors des luttes locales de territoire. La taille du contexte dépend de la taille du motif considéré, les plus utilisés sont ceux de taille 3x3 introduit par le programme Mogo [Gel+06] mais il est possible de considérer de plus grandes tailles. Ces motifs servent à simuler des luttes de territoire locale.

Définition 3.12: Schéma enrichi ou expert

Un schéma expert est un schéma partiel dont le contexte correspond à une représentation enrichie et partielle de l'environnement. La réponse indique l'action à jouer ensuite qui est en général étroitement liée à la représenta-

tion enrichie présente du contexte (cf. figure 3.8). Il s'agit de connaissances enrichies, floues et spatiales (pour l'essentiel).

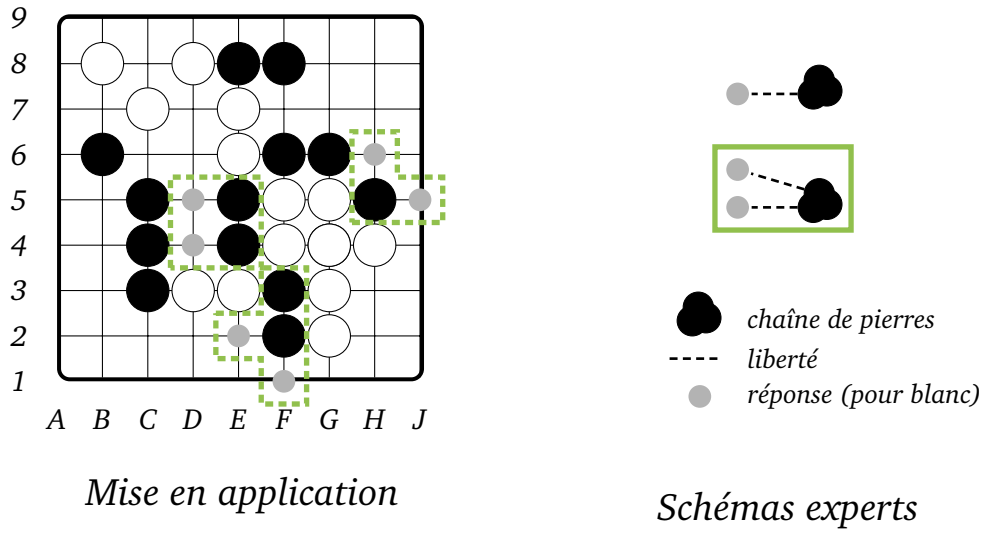


Figure 3.8.– Exemple de schéma expert : parmi les schémas experts présentés, seul le deuxième en partant du haut est applicable à la situation de jeu proposée. À chaque application, ce schéma suggère deux réponses possibles.

Les schémas experts sont équivalents à des règles expertes. Les représentations enrichies servent à capturer des propriétés spécifiques au jeu afin d'apporter à l'agent des connaissances tactiques élémentaires. Par exemple dans le jeu de Go, les schémas experts intègrent des notions spécifiques au jeu comme le nombre de libertés restantes d'une chaîne. De telles règles étaient fréquemment employées dans les programmes de Go avant l'introduction de MCTS[BC01]. Elles existent toujours dans les programmes avec MCTS[Enz+09] mais dans une moindre mesure.

3.1.2.3. Réseaux de Bradley-Terry

Les réseaux de Bradley-Terry [4] ont été introduits par Coulom pour le jeu de Go[Cou07a]. Cette structure permet d'agréger un ensemble de connaissances disparates au même niveau (cf. figure 3.9). Son fonctionnement repose sur le modèle de Terry-Bradley. Il s'agit du modèle à l'origine du classement Elo ; fréquemment utilisé pour comparer les joueurs dans des jeux comme les échecs ou le Go. À l'instar d'un joueur, chaque connaissance dispose d'un score et la combinaison des connaissances revient à com-

4. Dans la littérature cette représentation est présentée comme un modèle de Bradley-Terry. Nous avons choisi de le présenter comme un réseau de représentations par analogie à un réseau de neurones. Au cours d'une discussion sur la mailing-list computer-go[Com09], Coulom a d'ailleurs reconnu qu'il était possible de présenter sa méthode comme un réseau de neurones.

biner les scores comme pour les compétitions en équipe. Ces modèles sont utilisés dans de nombreux programmes pour établir les distributions de probabilités des actions [GP14 ; Shi11].

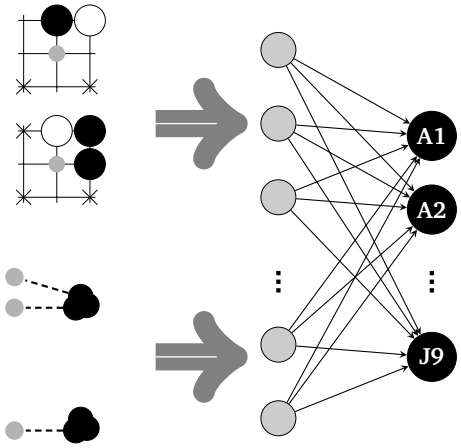


Figure 3.9.– Exemple de réseau de Bradley-Terry

Définition 3.13: Réseau de Bradley-Terry

Un réseau de Bradley-Terry est une structure qui calcule la probabilité de jouer un ensemble d'actions données à partir d'un ensemble de connaissances floues ou enrichies (cf. figure 3.9).

Les probabilités sont calculées à la volée suivant un modèle de Bradley-Terry. Chaque connaissance présente dans le réseau dispose d'un score. La probabilité de jouer une action est calculée à partir du score des connaissances vérifiées par l'action. Ces connaissances traitent aussi bien de la configuration du plateau autour de l'action (connaissances spatiales) que du dernier coup joué (connaissances temporelles).

Le modèle de Bradley-Terry autorise une grande souplesse dans son élaboration et son utilisation. En effet chaque connaissance est évaluée par un score v_i indépendamment des autres, c'est à dire sans avoir à considérer les différentes combinaisons possibles de connaissances. Lorsqu'une action vérifie plusieurs connaissances, la combinaison est calculée à la volée à partir des scores individuels. Par ailleurs cette structure sert en principe à établir la distribution de probabilités de l'ensemble des actions possibles mais peut très bien être restreinte à un sous-ensemble d'actions [Cou07a].

Limitation- Nécessité de connaissances abstraites Les combinaisons réalisées par le réseau de Bradley-Terry sont relativement simples en comparaison des réseaux de neurones. Ces relations ne se prêtent pas à comparer des connaissances brutes

et précises comme la position de chacune des pierres. Ainsi de tels réseaux sont contraints d'utiliser des connaissances abstraites (floues ou enrichies) pour pallier la simplicité de leur combinaison. Chaque connaissance floue ou enrichie introduit un biais dans l'évaluation. Le biais introduit par la conjonction de ces connaissances disparates devient plus difficile à caractériser.

3.1.2.4. Réseau de neurones

Un réseau de neurones artificiel est une structure de données inspirée par le fonctionnement du cerveau humain. Cette structure de données est propice à l'apprentissage de fonctions complexes difficiles à formaliser explicitement. Les réseaux de neurones artificiels ont notamment été employés avec succès pour des tâches d'identification comme la reconnaissance de formes par exemple[Lec+98]. Les réseaux de neurones ont été appliqués avec succès aux jeux comme par exemple au Backgammon [Tes95]. Le jeu de Go étant très visuel, les réseaux de neurones semblent propices à l'élaboration d'une fonction d'évaluation. Les réseaux de neurones convolutionnels sont appliqués au jeu de Go depuis plus de 20 ans[SDS94]. Toutefois, c'est par l'augmentation des capacités de calcul des ordinateurs et des techniques de programmation [5] que les réseaux de neurones convolutionnels ont récemment obtenu d'impressionnantes performances pour prédire la prochaine action à jouer [Mad+14 ; CS14]. Cependant les réseaux de neurones seuls ne suffisent pas à produire des programmes compétitifs en l'état actuel.

Définition 3.14: Réseau de neurones

Un réseau de neurones est une structure qui calcule la probabilité de jouer des actions à partir d'une représentation brute et précise d'un état de l'environnement. Cette structure est composée d'un ensemble de neurones reliés entre eux. Un neurone est une unité élémentaire capable d'émettre un signal lorsqu'il est suffisamment stimulé. Ces neurones sont connectés entre eux par des axones et organisés en couches successives. La couche d'entrée perçoit l'état du jeu et la couche de sortie donne pour chaque action une probabilité calculée à la volée.

Un neurone pris indépendamment ne possède aucune signification intrinsèque. Ce sont le nombre de neurones, la manière dont ils sont connectés aux autres neurones et le poids des axones qui détermineront ce que le réseau aura appris.

5. Le calcul massivement parallèle sur carte graphique a permis de traiter un plus grand nombre de neurones. Les réseaux obtenus comportent à présent plus de 5×10^5 neurones. À titre comparatif, le cerveau humain en comporterait environ 1×10^{11}.

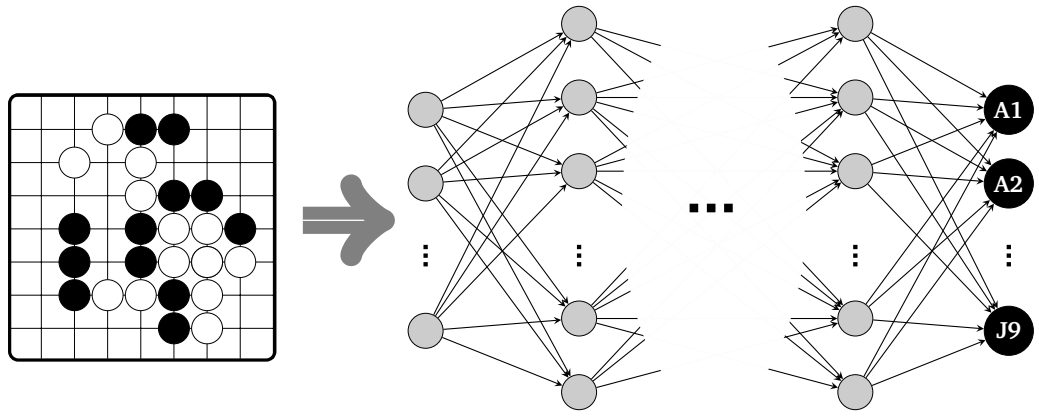


Figure 3.10.– Réseau de neurones

Lorsqu'une représentation brute et précise est fournie en entrée du réseau, la position des pierres stimule les premiers neurones ((cf. figure 3.10)). Leur signal se répercute de couches en couches jusqu'à atteindre la couche de sortie. À mesure que les signaux se propagent, le réseau réalise implicitement une forme d'abstraction non formalisée. C'est pourquoi, un réseau de neurones ne requiert, en principe, que la disposition actuelle des pièces pour calculer la distribution de probabilité (représentation brute et spatiale). Néanmoins certaines contributions suggèrent de tenir aussi compte de la dernière action jouée [CS14] (représentation temporelle) pour faciliter l'évaluation ou bien de réaliser des traitements spécifiques de la représentation brute pour certaines couches [Enz03] (représentation enrichie).

Limitation - Boîte noire Comme nous l'avons évoqué, les réseaux abstraient progressivement la représentation lors de la propagation du signal mais ces abstractions ne sont pas clairement identifiées pour l'heure. Le réseau de neurones opère à l'image d'une boîte noire à ce niveau : il n'est pas possible, en l'état actuel, d'identifier des zones spécifiques à certaines tâches. Toutefois les connections réalisées entre les neurones à l'instar des réseaux de neurones convolutionnels permettent l'émergence de certaines abstractions élémentaires comme des invariances par translation ou par symétrie [CS14].

3.1.3 Récapitulatif

Nous présentons la table 3.1 résumant les propriétés présentées dans la section 3.1.1 pour chacune des connaissances couvertes dans cet état de l'art. Les méthodes classiques d'IA appliquées aux jeux comme MinMax ou MCTS nécessitent de couvrir avec précision les états proches de la situation courante. Par conséquent ces méthodes reposent sur une représentation en arbre de recherche. Les différents types de

Représentation	Dimension	Traitement	Couverture
Arbre de recherche	Temporelle (et Spatiale)	Brute	Précise
Schéma spatial	Spatial	Brute	Floue
Schéma temporel	Temporel	Brute	Floue
Schéma expert	Spatial	Enrichie	Floue
Réseau de Bradley-Terry	Spatial	Enrichie	Floue
Réseau de Neurones	Spatial (et Temporel)	Brute (et Enrichie)	Précise

Table 3.1.– Récapitulatif des représentations dans le domaine des jeux

schémas partiels ainsi que les réseaux de Bradley-Terry sont généralement employés comme heuristique complémentaire ; afin d'améliorer l'évaluation des états présents dans l'arbre de recherche ou bien, dans MCTS, orienter les simulations aléatoires. Les réseaux de neurones, quant à eux, ont aussi bien été utilisés pour évaluer des états d'un arbre de recherche à l'instar du programme de Backgammon TD-Gammon[Tes95], mais peuvent aussi être utilisés tels quels pour décider de la prochaine action comme c'est généralement le cas pour le Go[Enz03 ; Mad+14]. Malgré les avancées obtenues par l'accroissement des puissances de calcul, les performances obtenues par des réseaux de neurones seuls sont encore inférieures à celles obtenues par des méthodes classiques.

3.2 Apprentissage dans le domaine des jeux

Dans cette section, nous allons à présent détailler les différentes méthodes d'apprentissage dans le domaine des jeux. Le terme d'apprentissage renvoie à la conception de connaissances pour le problème. Cela ne signifie pas nécessairement que l'agent participe à leur élaboration. Suivant la méthode d'apprentissage, l'agent disposera d'une plus grande autonomie dans l'acquisition de ses représentations. Nous couvrirons ici les trois méthodes d'apprentissage ci-dessous dans chacune des prochaines sections :

— Apprentissage expert ;

— Apprentissage supervisé ;

— Apprentissage par renforcement.

Chacune des méthodes est issue de courants différents de l'intelligence artificielle. Néanmoins les représentations issues de chacune de ces trois méthodes sont souvent utilisées conjointement dans les programmes de jeu compétitifs. À titre d'exemple les représentations issues de ces trois méthodes sont actuellement utilisées dans les

programmes de Go. Afin de les distinguer les unes des autres nous avons choisi de les traiter suivant trois critères distincts :

— Méthode d'apprentissage ;
— Représentations apprises ;
— Dynamique d'apprentissage.

Les deux premiers critères renvoient notamment aux deux axes d'analyses identifiées dans le chapitre 2. Les représentations apprises indiquent quelles sont les représentations parmi celles présentées dans la section précédente qui sont conçues ou évaluées à partir de cette méthode. Il s'agit de définir quel niveau d'abstraction ces méthodes sont susceptibles de traiter (cf. section 2.2.1). La méthode d'apprentissage indique notamment les informations utilisées pour concevoir ses représentations internes de l'agent. Pour des méthodes d'apprentissage automatisé comme l'apprentissage supervisé ou l'apprentissage par renforcement, il s'agit de définir les rétroactions ainsi que leur exploitation par l'agent (cf. section 2.2.2).

Enfin, la dynamique d'apprentissage précise comment vont évoluer ces représentations au cours de l'apprentissage (cf. Définition 1.3 p. 10). Il s'agit de considérer les aspects dynamiques dans l'apprentissage et comment elles sont mises en oeuvre. Parmi ces trois méthodes d'apprentissage, l'apprentissage par renforcement est la seule à faire intervenir le processus de résolution au cours de l'apprentissage. Nous présenterons en particulier les implications sur la dynamique d'apprentissage.

Un tableau récapitulatif résumera pour chacune des méthodes les principaux critères en fin de section.

3.2.1 Apprentissage expert

Le développement des connaissances expertes renvoie aux débuts de l'intelligence artificielle. L'enjeu était alors de réussir à encoder les connaissances d'un expert du domaine (comme un joueur professionnel) sous une forme numérique. Ce type d'apprentissage repose sur des motifs ad-hoc ou des schémas experts [Bou05]. De telles connaissances exploitent en général des spécificités du problème afin d'en tirer des heuristiques de recherche. Ces représentations sont ensuite intégrées à l'agent qui en dispose comme des connaissances innées : il ne participe à aucun moment à leur conception.

L'avancement en intelligence artificielle est tel qu'il est encore indispensable d'exploiter les spécificités du problème afin de produire des programmes compétitifs pour des jeux combinatoires difficiles comme le Go ou les échecs. Dans des jeux

comme le Go, les programmes ont longtemps employé un grand nombre de représentations expertes afin d'intégrer des raisonnements semblables à ceux de joueurs humains [BC01]. Il se trouve d'ailleurs que les concepteurs du joueur artificiel sont eux-mêmes des spécialistes du jeu en question.

L'agent n'a aucune prise sur ces représentations qui lui sont fournies *a priori*. Il ne peut pas en principe les remettre en question : elles sont par définition statiques du point de vue de l'agent.

3.2.1.1. Méthode d'apprentissage

Ces représentations sont essentiellement jugées par leur concepteur. C'est son expertise du jeu qui lui permet d'identifier les informations les plus intéressantes du jeu. L'impact en pratique de ces représentations sur un joueur artificiel est jugé lors de compétitions entre programmes. À méthodes équivalentes, ce sont généralement ces représentations qui feront la différence face à d'autres compétiteurs. Les performances des programmes commerciaux de Go sont vraisemblablement dues aux représentations expertes supplémentaires, de l'aveu même de leur concepteur [6].

Lors de leur conception, l'expert peut de lui même identifier le moment du jeu où il est préférable d'utiliser ces représentations. Par exemple les livres d'ouverture à nouveau sont dédiés aux premiers tours du jeu. Cependant au-delà des premiers tours, il devient difficile pour un expert de définir *a priori* le meilleur moment pour les utiliser. C'est pourquoi elles servent le plus souvent à établir des heuristiques vérifiées tout au long de la partie.

3.2.1.2. Représentations traitées

Les contributions des experts portent avant tout sur la conception des représentations. En particulier les représentations enrichies nécessitent de savoir identifier les informations pertinentes pour un jeu spécifique. De nombreuses représentations enrichies ont été développées à l'origine par des joueurs du jeu comme les *Common Fate Graph* ou bien les schémas experts [Enz+09].

La quantité de représentations expertes est de fait limitée par les capacités de l'expert et par conséquent relativement restreinte. L'enjeu est alors de fournir à l'agent des règles générales qu'il pourra appliquer dans de nombreuses situations. Ainsi ces représentations sont généralement floues : c'est le cas par exemple des

6. Il n'existe cependant aucun article scientifique détaillant le fonctionnement de ces programmes. Ces informations reposent sur des communications personnelles dans la mailing-list computer-go.

schémas experts ou des schémas spatiaux. Dans certains rares cas cruciaux, ces représentations sont précises : c'est le cas par exemple des livres d'ouvertures en début de partie [7].

3.2.1.3. Dynamique d'apprentissage

Les représentations sont fournies *a priori* à l'agent. Elles restent statiques tout au long de la partie. Il n'y a donc aucune dynamique d'apprentissage de point de vue de l'agent. D'une partie à l'autre ces représentations sont toutefois susceptibles d'évoluer mais ce sera à l'initiative de l'expert. La dynamique d'apprentissage existe alors dans une certaine mesure pour l'expert. Il s'agit d'une dynamique longue mais qui peut s'avérer fructueuse. Les schémas spatiaux du programme Indigo[Bou05] ont été ainsi progressivement ajustés au cours de compétitions de Go. Ces schémas ont notamment servi à élaborer le programme Mogo[Gel+06] (le premier à atteindre un professionnel de jeu).

3.2.2 Apprentissage supervisé

Dans l'apprentissage supervisé, l'agent apprend de lui-même à partir d'un ensemble d'exemples qui lui sont fournis. Pour chaque exemple, une mise en situation est fournie accompagnée de la réponse attendue. Cet apprentissage est supervisé car ces réponses ont été établies au préalable par un expert qui, à l'instar d'un professeur, indique à l'apprenant la bonne réponse. L'agent doit alors synthétiser les connaissances présentes dans l'ensemble des exemples fournis.

Ces méthodes reposent sur des modèles statistiques prédictifs (Bayésien[SHG06], Bradley-Terry[Cou07a]) pour tirer de grands échantillons de données des enseignements. C'est pourquoi il est essentiel de disposer d'un grand nombre d'exemples. Avec l'essor des volumes de données, l'apprentissage supervisé devient de plus en plus important dans le domaine des jeux mais aussi dans de nombreux autres domaines de l'IA.

Les représentations acquises sont progressivement ajustées par l'agent au cours de l'apprentissage de façon automatique. Néanmoins cet apprentissage repose énormément sur les données fournies à l'agent. La réponse attendue est tout d'abord établie *a priori* par un expert ou dans la mise en situation proposée. L'agent n'est pas force de proposition dans aucun de ces cas.

7. Les livres d'ouvertures sont des tables qui indiquent en début de partie les actions à répondre à l'adversaire. Les premières actions jouées sont souvent les mêmes dans les jeux combinatoires car elles sont unanimement jugées comme meilleures aux autres. Une entorse aux réponses classiques est souvent lourde de conséquences sur la suite de la partie.

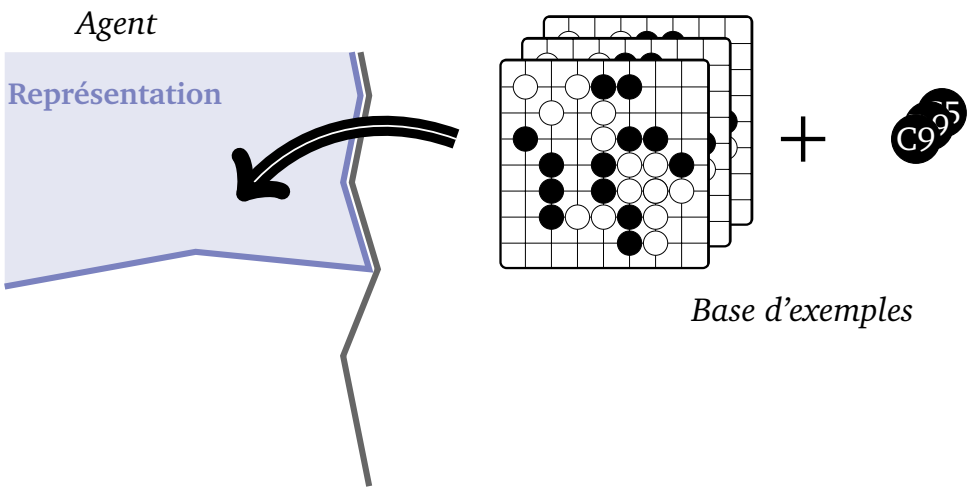


Figure 3.11.– Decription de l'apprentissage supervisé : l'agent ajuste ses représentations par la seule observation d'une base d'exemples qui lui sont fournis. Le processus de résolution ne participe pas à l'apprentissage.

3.2.2.1. Méthode d'apprentissage

Les représentations sont évaluées en confrontant successivement à l'agent des nombreuses situations différentes regroupées dans une base d'apprentissage. À chacune de ces situations, une réponse en particulier est attendue de l'agent. La réponse fournie par l'agent est comparée à celle attendue. Les évaluations des représentations sont ensuite ajustées de manière à correspondre le mieux avec la réponse attendue dans l'exemple fourni. Les représentations sont jugées à partir des réponses attendues, qui sont fournies au préalable à chacune des situations. Ces exemples constituent des représentations externes à l'agent qui vont lui servir à ajuster ses propres représentations.

La principale difficulté pour ce genre d'approche est de disposer d'une base d'apprentissage suffisamment grande. En effet, il est nécessaire qu'un tuteur externe identifie pour chaque situation la réponse qui était attendue (à l'instar d'un professeur). Il s'agit d'une tâche longue et fastidieuse même pour un joueur averti. Toutefois, dans des jeux populaires comme le Go, il existe de nombreux enregistrements de parties entre professionnels. De tels enregistrements peuvent être facilement transformés en une vaste base d'apprentissage. Pour chaque état atteint au cours d'une partie, la réponse attendue correspond à celle effectivement jouée par le joueur ensuite. Une même partie fournit alors autant de situations qu'il y a eu de tours au cours de la partie.

Ainsi les représentations sont évaluées à partir d'actions réalisées par d'autres joueurs jugés meilleurs que l'agent. Par ailleurs ces représentations sont ajustées à partir

de l'action immédiatement réalisée ensuite et non à partir d'objectifs de plus long terme comme l'issue de la partie. Il s'agit d'acquérir des techniques communément appliquées dans les jeux entre professionnels. La qualité de l'apprentissage dépend principalement du modèle prédictif retenu.

3.2.2.2. Représentations traitées

L'apprentissage supervisé sert principalement à ajuster les évaluations ou les poids des représentations. Les représentations à évaluer sont alors fournies *a priori* au programme. Comme l'apprentissage est automatisé, il est employé en général pour évaluer un grand nombre de représentations brutes différentes comme des schémas spatiaux[SHG06] ou bien des représentations avec de nombreux poids à ajuster comme les réseaux de neurones[Mad+14 ; CS14] ou les réseaux de Bradley-Terry[Cou07a ; GP14].

Cet apprentissage automatisé vise à capitaliser des représentations abstraites sur le jeu à l'instar des représentations expertes. Ainsi un tel apprentissage s'adresse généralement à des représentations abstraites comme les schémas spatiaux (floue) ou les réseaux de Bradley-Terry (enrichie). Cependant des représentations adéquates comme les réseaux de neurones sont susceptibles d'intégrer une telle abstraction dans leurs évaluations internes alors qu'il s'agit d'une représentation précise.

3.2.2.3. Dynamique d'apprentissage

Bien que les évaluations soient ajustées automatiquement, l'apprentissage reste très encadré : aussi bien les situations rencontrées que les réponses sont fournies par un tuteur externe. Par conséquent il n'y a aucun ajustement au cours de l'apprentissage : l'ordre suivant lequel sont présentées les différentes situations ne s'adapte pas aux représentations actuellement acquises par le système, aucune progressivité de l'apprentissage n'est envisagée. L'apprenant est complètement passif vis à vis de son apprentissage : il n'en contrôle ni le rythme ni la teneur.

L'apprentissage supervisé cherche à définir des représentations d'ordre général sur le jeu. Il suffit de les établir une fois pour toute à partir de la base d'apprentissage fournie. Par conséquent l'apprentissage est réalisé hors-ligne et indépendamment d'un processus d'exploitation en particulier. Une fois apprise, ces représentations demeureront statiques (même lorsqu'elles seront intégrées à un agent).

3.2.3 Apprentissage par renforcement

L'apprentissage par renforcement est une méthode d'apprentissage par essai et erreur. L'agent est plongé dans l'environnement et reçoit des renforcements positifs ou négatifs suivant les actions qu'il réalise. Les renforcements sont définis conformément à un objectif fixé au préalable pour le problème. À l'instar des méthodes d'apprentissage supervisé, les renforcements sont définis *a priori* par un tuteur externe à l'agent. En revanche, c'est à l'agent d'interpréter, à partir des renforcements perçus, quelles sont les actions qui lui ont permis d'atteindre l'objectif.

Dans le cadre des jeux combinatoires, l'objectif est clairement défini *a priori* et il est facile d'établir la fonction de renforcement associée : un renforcement positif en cas de victoire et sinon un renforcement négatif. L'apprentissage par renforcement a été appliqué avec succès à ce domaine. Par le passé le programme TD-Gammon[Tes95] a atteint un niveau de jeu professionnel au backgammon grâce à l'apprentissage par différence temporelle TD(λ). Plus récemment les programmes de Go ont aussi atteint un niveau professionnel de jeu grâce aux méthodes de Monte Carlo.

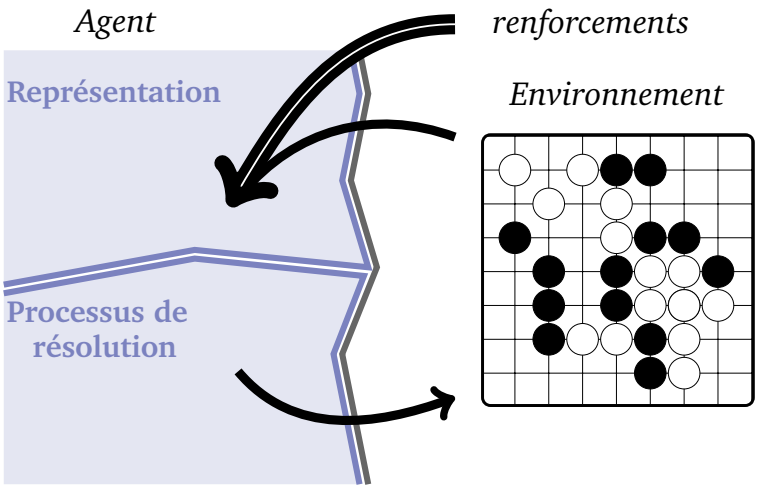


Figure 3.12.– Decription de l'apprentissage par renforcement : l'agent apprend par l'observation de renforcements consécutifs à ses actions. Le processus de résolution participe à l'apprentissage.

Comme dans l'apprentissage supervisé les représentations sont ajustées automatiquement. Cependant le processus de résolution de l'agent est partie prenante de l'apprentissage en choisissant de lui-même les prochaines actions qu'il va réaliser.

3.2.3.1. Processus de résolution

L'apprentissage par renforcement a été initialement conçu pour les Problèmes Décisionnels de Markov (PDM). La solution d'un PDM est un processus. Par conséquent, l'apprentissage porte en principe sur le processus appelé en l'occurrence une politique π, à la différence des autres méthodes présentées.

Définition 3.15: Politique

Une politique π est la distribution de probabilité des actions qu'un agent peut sélectionner depuis un état quelconque. Cette fonction associe à tout état s et toute action a une valeur comprise dans l'intervalle $[0;1]$: $\mathcal{S} \times \mathcal{A} \mapsto [0;1]$ telle que :

$$\forall s \in \mathcal{S}, \sum_{a \in \mathcal{A}} \pi(s,a) = 1$$

Une politique déterministe correspond à une politique où pour tout état s, il existe une unique action pour laquelle $\pi(s,a)$ est égale à 1 (la probabilité pour les autres actions est nulle). La politique est le processus d'un agent au sens de la définition 1.2 (p. 9) dans le cadre d'un problème décisionnel de Markov.

3.2.3.2. Méthode d'apprentissage

Les représentations sont évaluées au cours d'expériences réalisées par l'agent dans l'environnement. À chacune de ses actions, il perçoit des récompenses qui lui indiqueront dans quelle mesure il atteint l'objectif. Dans le cadre de jeux combinatoires, chaque partie représente une expérience distincte habituellement appelée épisode en apprentissage par renforcement. Chaque épisode démarre du même état initial et se déroule en un temps fini de tours de jeu T. La seule récompense perçue par l'agent r_T survient en fin d'épisode lorsque le résultat de la partie est identifiable sans aucune ambiguïté. De façon plus formelle, une partie entre deux joueurs e^j est la séquence des états s_i atteints et actions a_i réalisées à chaque tour :

$$e^j = (s_0^j, a_0^j, s_1^j, a_1^j, \ldots, a_{T-1}^j, s_T^j, r_T^j)$$

La définition de la récompense représente la seule influence externe à l'agent, à la différence de l'apprentissage supervisé où l'état et la réponse attendue sont fournis à l'agent. L'agent va apprendre à partir de cette seule récompense et de sa propre

expérience. La récompense perçue va servir à établir les évaluations associées à chacun des états traversés au cours de l'épisode. Le cours de la partie est étroitement lié au processus π qui a servi à générer cet épisode. C'est pourquoi contrairement aux évaluations réalisées par d'autres formes d'apprentissage, les valeurs d'actions sont dépendantes de la politique d'apprentissage.

3.2.3.3. Représentations traitées

L'apprentissage d'une politique nécessite en général d'accumuler des représentations sur le problème. Les représentations apprises portent sur l'évaluation de représentations brutes et précises des états de l'environnement comme les fonctions de valeur d'action. Les preuves d'optimalité de la méthode ont notamment été établies pour de telles représentations[SB98].

Définition 3.16: Fonction de valeur d'action

Une fonction de valeur d'action Q^π (*Action value function* en anglais) évalue un couple état-action suivant une politique π. Cette valeur représente les récompenses que l'agent espère percevoir s'il applique la politique π depuis cet état. Lorsque la récompense ne survient qu'en fin d'épisode, la fonction de valeur d'action est définie telle que :

$$Q^\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi \left\{ \gamma^{T-t} r_T | s_t = s, a_t = a \right\}$$

avec r_T la récompense perçue en fin d'épisode et $\gamma \in [0;1]$, le taux d'amortissement (*discount rate* en anglais). Le paramètre γ sert à établir un compromis entre les récompenses survenant à longue échéance de celles plus immédiates pour un même couple état-action. Dans les jeux combinatoires, cette valeur peut être interprétée comme la probabilité que la partie se poursuive étant donné la situation courante [Lit94]. Par défaut, il sera fixé à 1.

Par le passé, les méthodes d'apprentissage par renforcement ont été aussi utilisées notamment pour ajuster les poids de réseaux de neurones [Enz03 ; Tes95]. À défaut de disposer d'une base d'exemples suffisamment grande pour réaliser un apprentissage supervisé, l'apprentissage par renforcement permet d'apprendre par *self-play* : c'est à dire en se confrontant à lui-même dans plusieurs parties. Plus récemment, les méthodes à base de simulations de Monte Carlo évaluent un arbre de recherche au cours de la partie [Bro+12 ; Caz09]. La plus emblématique d'entre elles est Monte Carlo Tree Search sur laquelle nous reviendrons dans le chapitre suivant.

Abstraction des représentations Les méthodes d'apprentissage par renforcement ont toutefois été aussi appliquées pour des représentations plus abstraites. Bien qu'il n'existe alors plus aucune garantie de performance, les résultats obtenus en pratique peuvent s'avérer néanmoins remarquables. Le programme de Go RLGO [SSM07 ; SSM12] par exemple a appris à évaluer un ensemble de schémas spatiaux (et donc flous). Bien que les performances de RLGO ne rivalisent pas avec des méthodes de Monte Carlo, les évaluations des schémas obtenues par RLGO ont par la suite été intégrées avec succès à des méthodes de Monte Carlo[GS07]. Toutefois, si la représentation choisie est trop simpliste, la méthode risque alors de ne pas converger comme cela est illustré dans le mémoire de Lew[Lew11].

Des représentations floues peuvent aussi être évaluées par renforcement, en complément de représentations plus précises (comme un arbre de recherche par exemple). C'est le cas notamment des schémas spatiaux[Bas+14] et temporels[BD10 ; TWB12] utilisés pour renforcer MCTS.

3.2.3.4. Dynamique d'apprentissage

Dans l'apprentissage par renforcement, processus de résolution (π) et représentation (Q^π) interviennent successivement au cours de l'apprentissage de l'agent. Politique et valeurs d'états s'influencent mutuellement. Lors de l'évaluation de la politique, la politique modifie les valeurs d'états tandis que les valeurs d'états modifient la politique lors de l'amélioration de la politique. Cette dynamique est commune à l'ensemble des méthodes d'apprentissage par renforcement sous le nom d'Itération de la Politique Généralisée (*Generalized Policy Iteration* en anglais).

Définition 3.17: Itération de la Politique Généralisée (IPG)

L'itération de la politique généralisée approxime une politique maximale par modifications progressives d'une politique initiale et d'une fonction de valeur d'action arbitraire. Cet algorithme général est composé de deux phases successivement répétées :

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{A} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{A} \pi_1 \xrightarrow{E} \dots \xrightarrow{A} \pi^* \xrightarrow{E} Q^*$$

- Évaluation de la politique $\xrightarrow{E}$: les valeurs Q^π sont calculées par l'application (répétée ou non) de la politique π ;
- Amélioration de la politique $\xrightarrow{A}$: une nouvelle politique π' est construite à partir des valeurs Q^π afin de se rapprocher de la politique maximale π^*.

L'itération de la politique généralisée ne précise pas plus les détails suivant lesquels chacune des deux phases sont réalisées. Il s'agit d'une description générale des interactions entre une politique π et une fonction de valeur d'action Q^π au cours du temps qu'il est possible de schématiser sous une boucle comme présenté dans la figure 3.13 :

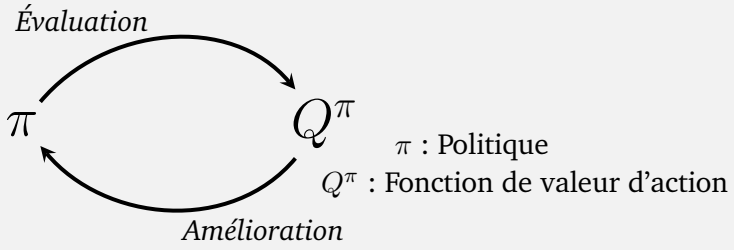


Figure 3.13.– Schéma de l'IPG

L'IPG converge asymptotiquement vers la politique maximum d'après l'équation de point fixe de Bellman. Cependant cette convergence implique d'employer des représentations précises de l'environnement d'une part et de parcourir l'ensemble des états de l'environnement d'autre part : ce qui n'est jamais le cas (sauf dans les cas simples). Il s'agit néanmoins d'un cadre descriptif qui nous permet de mettre en avant les interactions entre processus de contrôle et représentations. Ces interactions se découpent en améliorations de la politique et évaluations de la politique. Leur mise en oeuvre est décisive de la réussite en pratique de l'apprentissage. Nous détaillons plus en avant par la suite chacune de ces interactions et, en particulier, pour les méthodes de Monte Carlo.

Amélioration de la politique L'amélioration de la politique modifie la politique à partir des valeurs d'actions calculées de manière à se rapprocher d'une politique maximale. Cependant la politique reste approximative au cours des premières itérations. L'amélioration de la politique [8] ne peut pas reposer sur les seules valeurs d'états car celles-ci ne concernent que les états qui ont été couverts jusqu'à présent et ne tiennent pas compte de tous ceux qui n'ont pas été encore explorés. Ainsi des états sous-optimaux peuvent s'avérer bien meilleurs pourvu que la politique se risque à les essayer de nouveau. Tout l'enjeu repose sur l'importance qu'il faut accorder aux autres états par rapport à ceux actuellement jugés comme meilleurs. Ce dilemme est identifié dans la littérature comme le compromis entre exploration et exploitation.

8. Dans la littérature les politiques se distinguent entre elles par leur phase d'amélioration de politique associée. Par abus de langage, la phase d'amélioration de politique est souvent identifiée à la politique même.

Définition 3.18: Compromis exploration-exploitation

Le compromis entre exploration et exploitation représente la propension d'une politique d'apprentissage à essayer des actions considérées comme sous-optimales (exploration) au profit d'actions considérées comme optimales (exploitation) étant donné les évaluations actuelles.

L'exploitation privilégie les actions optimales afin de maximiser les renforcements immédiats tandis que l'exploration sélectionne des actions sous-optimales dans l'espoir de maximiser les renforcements à plus long terme (cf. annexe A.2).

Ce compromis est généralement réalisé grâce à une politique stochastique inspirée de la politique gloutonne qui sélectionne systématiquement les actions avec les meilleures évaluations. Par exemple, la politique $\epsilon-$gloutonne sélectionne avec une probabilité de $1-\epsilon$ la meilleure action et une autre action aléatoire dans le cas contraire. La valeur ϵ indique dans quelle mesure la politique va explorer des états sous-optimaux. Il est aussi possible de définir des politiques déterministes qui autorisent l'exploration d'états sous-optimaux comme la politique *Upper Confidence Bound* (UCB) développée pour les bandit à n-bras (cf. annexe A.2).

Évaluation de la politique L'évaluation d'une politique est aussi appelée le problème de la prédiction (*prediction problem*). En effet cette phase revient à prédire les gains espérés en suivant la politique courante. Les évaluations des actions ne sont pas calculées au cours d'une seule phase. Elles sont ajustées progressivement tout au long de l'IPG à chaque fois que l'état a été observé dans un épisode. Le calcul des évaluations est réalisé incrémentalement par renforcements successifs. Pour toute méthode d'apprentissage par renforcement, l'évaluation de la politique suit systématiquement le motif suivant :

$$\textit{Nouvelle éval.} \longleftarrow \textit{Ancienne éval.} + \alpha \left[\underbrace{\textit{Observation cible} - \textit{Ancienne éval.}}_{\delta} \right]$$

Le paramètre α représente le taux d'apprentissage. Il définit à quelle rythme l'agent va apprendre de ses expériences. En principe, sa valeur décroît à mesure que l'estimation devient de plus en plus précise ; afin d'assurer la convergence des évaluations [SB98] [9]. La quantité δ représente l'erreur commise par l'estimation courante. Une erreur d'estimation par différence temporelle est la différence entre l'estimation d'une valeur à un instant donné de l'épisode et une observation cible obtenue plus

9. Le taux d'apprentissage est laissé constant pour des problèmes non-épisodiques (à horizon infini) ou lorsque l'environnement est susceptible d'évoluer de lui-même au cours de l'apprentissage.

tard au cours du même épisode. La nature précise de l'observation cible varie d'une méthode d'apprentissage par renforcement à une autre.

En particulier l'évaluation de la politique dite de Monte Carlo utilisent exclusivement les récompenses finales comme observation cible : les erreurs commises par les estimations δ sont calculées par rapport à la récompense finale de l'épisode r_T. L'évaluation de la politique suit le schéma suivant (avec $\gamma = 1$) :

$$Q^{\pi}_{k+1}(s,a) = Q^{\pi}_{k}(s,a) + \underbrace{\frac{1}{k+1}}_{\alpha_k} \left[\underbrace{r_T - Q^{\pi}_{k}(s,a)}_{\delta} \right] \tag{3.1}$$

L'ajustement réalisé est équivalent au calcul incrémental d'une moyenne statistique : $Q^{\pi}_{k}(s,a)$ représente la valeur moyenne actuelle, l'indice k correspond à la taille de l'échantillon et r_T est la nouvelle valeur à ajouter. En l'occurrence, le taux d'apprentissage α_k dépend du nombre de mise à jour et décroît à mesure que celui-ci augmente (garantissant la convergence de l'évaluation).

L'erreur est calculée à partir du renforcement final exclusivement. Ainsi cette méthode ne bénéficie pas des valeurs des autres estimations afin d'amorcer l'apprentissage (*bootstrap* en anglais) comme les évaluations de politiques TD(λ). À l'inverse, elles autorisent à n'estimer qu'un sous-ensemble des états couverts par les épisodes. Pour les états qui ne sont pas couverts par la politique courante π, les actions sont alors générées par une autre politique (aléatoire en général). Cette propriété est très intéressante pour les jeux combinatoires. En effet, le vaste espace d'états de ces jeux empêche toute évaluation exhaustive des états rencontrés lors de l'épisode. D'autre part, le seul renforcement précis est accessible en fin de partie (qui marque la fin de l'épisode).

3.2.4 Récapitulatif

La table 3.2 détaille les critères présentés en début de section pour chacune des trois méthodes couvertes.

Le choix de la méthode d'apprentissage dépend de la liberté que l'on souhaite laisser à l'agent pour élaborer ses propres connaissances. L'acquisition de connaissances expertes est la transcription des connaissances du concepteur à un joueur artificiel. L'agent ne peut pas remettre en question les représentations ainsi acquises. Dans l'apprentissage supervisé, l'agent reste guidé par la base d'exemples mais il établit de lui-même les évaluations. Il apprend par la seule observation des parties entre joueurs professionnels. Il synthétise en quelque sorte le comportement de tous ces joueurs

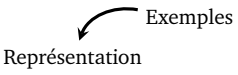
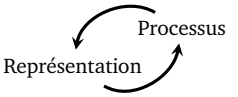
Méthode d'apprentissage	Représentations apprises	Méthode d'apprentissage	Dynamique d'apprentissage
Apprentissage expert	*Schémas spatiaux* [Bou05] *Schémas experts* [Enz+09]	Expert	Représentation
Apprentissage supervisé	*Schémas spatiaux* [SHG06], *Réseaux Bradley-Terry* [Cou07a ; GP14], *Réseaux de neurones* [CS14 ; Mad+14]	Base exemples	
Apprentissage par renforcement	*Arbre de recherche* [Bro+12 ; Caz09], *Schémas spatiaux* [SSM07 ; Bas+14], *Schémas temporels* [TWB12 ; BD10], *Réseaux de neurones* [Enz03 ; Tes95]	Renforcement	

Table 3.2.– Récapitulatif des méthodes d'apprentissage dans le domaine des jeux

professionnels mais il n'est pas en mesure de proposer une autre façon de jouer que celle observée. À l'inverse, l'apprentissage par renforcement intègre le processus de résolution dans son apprentissage. Ce qui lui permet notamment d'apprendre de lui-même. Des exemples par le passé ont montré que les joueurs artificiels apprenant par renforcement étaient capables de proposer une autre manière de jouer que celle pratiquée par les joueurs professionnels [10].

Lorsque les connaissances se formalisent bien pour le problème, il est plus intéressant de limiter la liberté de l'agent. Il devient en effet inutile de laisser l'agent apprendre de lui-même des connaissances déjà existantes par ailleurs : c'est le cas des connaissances expertes. Cependant lorsque le concepteur lui-même ne sait pas quelles sont les connaissances pertinentes ou comment formaliser ces connaissances, les méthodes d'apprentissage automatisé sont à privilégier. L'apprentissage supervisé est nécessairement réalisé en dehors de toute partie (hors-ligne). Le concepteur doit alors correctement identifier les bonnes représentations à évaluer afin qu'elles puissent couvrir toutes les situations auxquelles l'agent est susceptible d'être confronté : l'agent apprend des connaissances générales avec le risque d'introduire un biais dans certains cas. L'apprentissage par renforcement peut être accompli au cours d'une partie, à l'instar de MCTS, et permet d'apprendre des connaissances spécifiques à la situation actuelle : l'agent apprend alors des connaissances précises au risque de les apprendre de nouveau à chaque partie.

10. Pour le Backgammon par exemple, le programme TD-Gammon a ravivé une ouverture de partie alors négligée par joueurs professionnels [Tes95].

3.3 Synthèse

Dans ce chapitre nous avons traité des représentations employées dans le cadre des intelligences artificielles appliquées aux jeux combinatoires. Afin de pouvoir mieux distinguer les représentations entre elles, il nous a fallu distinguer plus en avant les représentations aux travers des propriétés que nous avons définies :

— Dimension ;

— Traitement ;

— Couverture.

Ces propriétés caractérisent notamment l'usage de ces représentations. Nous avons pu alors couvrir les principales représentations employées dans les jeux combinatoires en détaillant à chaque fois leurs limitations intrinsèques. Cette analyse est résumée dans le tableau 3.1.

Nous avons ensuite présenté les différentes méthodes pour concevoir ces représentations dans l'IA appliquée aux jeux. À nouveau nous avons couvert les principales méthodes d'apprentissage au travers de trois critères :

— Représentations apprises ;

— Méthode d'apprentissage ;

— Dynamique.

Cette analyse est résumée dans le tableau 3.2. Tous les critères ne sont pas explicitement traités par les différentes méthodes traitées mais ils nous permettent de mettre en avant les spécificités de l'apprentissage par renforcement. En particulier, cette méthode d'apprentissage dispose d'une dynamique d'apprentissage beaucoup plus riche que les autres méthodes : l'itération de la politique généralisée (cf. la définition 3.17). Cette dynamique nous servira comme support d'analyse par la suite.

La méthode de Monte Carlo Tree Search appartient à la famille des méthodes d'apprentissage par renforcement. Dans le chapitre suivant nous précisons encore plus l'état de l'art pour cette méthode d'apprentissage en particulier.

4 Monte Carlo Tree Search dans les jeux

La méthode Monte Carlo Tree Search (MCTS) a été introduite en 2006 dans le domaine des jeux[Cou07b]. Elle fut initialement appliquée avec succès pour le jeu de Go et est devenue une méthode classique de l'IA appliquée aux jeux, au même titre que la méthode $\alpha - \beta$. Cette méthode a fait l'objet de nombreuses contributions, notamment pour le jeu de Go. Dans les programmes actuels, de nombreuses améliorations sont combinées. À la différence des autres méthodes appliquées dans le domaine des jeux, MCTS est une méthode d'apprentissage à part entière, qui apprend des représentations pendant la partie. Il est alors plus difficile d'identifier les implications effectives de chacune des améliorations sur le comportement de l'agent, d'autant plus lorsqu'elles sont combinées.

Dans ce chapitre, nous réalisons un état de l'art spécifique à cette méthode dans le domaine des jeux. Nous présentons tout d'abord le fonctionnement général de la méthode canonique dans la section 4.1. Nous couvrons ensuite les principales améliorations apportées dans la littérature dans la section 4.2. Enfin nous positionnons chacune de nos contributions suivant l'état de l'art présenté en section 4.3.

4.1 Description canonique de Monte Carlo Tree Search

Monte Carlo Tree Search appartient à la famille des méthodes d'apprentissage par renforcement (cf. section 3.2.3). À l'image des méthodes de Monte Carlo, l'agent va ajuster ses représentations à partir des résultats de simulations aléatoires. Dans MCTS, l'agent va accumuler les renforcements dans un arbre de recherche et ainsi progressivement affiner sa connaissance du problème.

Dans cette section, nous présentons la méthode « canonique ». C'est à dire celle commune à toutes les applications qui ont été réalisées. La section 4.1.1 décrit tout d'abord le déroulement général de la méthode. Nous reviendrons ensuite sur chacune des 4 phases importantes de l'algorithme d'apprentissage dans la section 4.1.2. Ces phases nous permettront de positionner chacune des contributions de l'état de

l'art. Enfin nous exposerons dans la section 4.1.3, les différentes interactions entre chacune de ces phases, au travers de la dynamique d'apprentissage de la méthode.

4.1.1 Description générale de l'apprentissage dans MCTS

Dans cette section nous décrivons le déroulement de l'apprentissage réalisé par l'agent. Dans un premier temps, nous présentons le principe général de MCTS. Nous y évoquons notamment la nature des connaissances apprises par MCTS. Puis nous survolons le déroulement global d'un tour de jeu pour un joueur artificiel utilisant MCTS. L'apprentissage est réalisé par tâches épisodiques que nous décrivons plus en détail dans une troisième partie. Enfin nous reviendrons sur les principales variantes autour de MCTS.

4.1.1.1. Principe général

MCTS apprend par renforcement les évaluations d'états précis de l'environnement. Des milliers de fins de parties sont simulés depuis la position courante. Ces fins de parties représentent autant d'épisodes simulés par l'agent. Les récompenses obtenues suite à chaque épisode, sont progressivement accumulées dans un arbre de recherche (cf. Définition 3.7). Dans le même temps, l'arbre de recherche est graduellement étendu à partir de ces évaluations calculées à la volée. Ainsi l'apprentissage de l'agent est double : d'une part il renforce progressivement ses évaluations et d'autre part il accroît l'étendue de ses connaissances.

4.1.1.2. Description d'un tour de jeu

La quantité et la qualité des connaissances apprises par l'agent dépendent du nombre de renforcements perçus. Par défaut, l'agent se contente alors de générer le maximum d'épisodes possibles dans un tour de jeu. Le nombre d'épisodes est généralement contraint par une limite de temps ou bien d'épisodes par tour. En fin de compte, un agent MCTS est en perpétuel apprentissage. Suivant cette perspective, il apparaît d'autant plus important de considérer la dynamique de cet apprentissage car elle est en permanence à l'œuvre. Ainsi l'agent sera toujours en mesure de proposer une action quelque soit le nombre d'épisodes générées : c'est pourquoi cette méthode est qualifiée de *any-time*.

4.1.1.3. Description d'un épisode

Un épisode représente le déroulement d'une fin de partie simulée depuis la situation courante jusqu'à un état final. Dans le cadre de MCTS, un épisode est constitué de 4 phases ou étapes successives résumées dans la figure 4.1 que nous allons à présent détailler :

1. L'étape de *descent* ;
2. L'étape de *growth* ;
3. L'étape de *roll-out* ;
4. L'étape d'*update*.

L'étape de *descent* parcourt l'arbre et décide du prochain état à évaluer sur la base des évaluations stockées actuellement dans l'arbre. L'étape de *growth* ajoute cet état à l'arbre de recherche. À partir de cet état, les coups sont générés aléatoirement lors de l'étape *roll-out* jusqu'à atteindre une fin de partie où il devient facile d'évaluer le résultat. Ce résultat est ensuite rétro-propagé dans l'arbre de recherche lors de l'étape *update*.

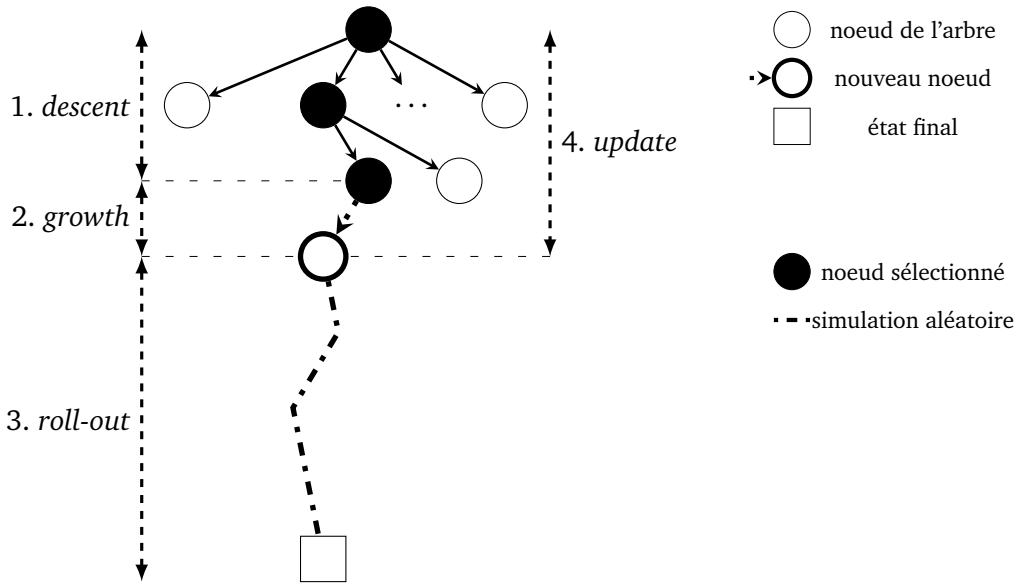


Figure 4.1.– Description de MCTS

Pour des jeux à deux joueurs, l'agent simulera aussi bien ses propres actions que celles de son adversaire suivant le principe de l'algorithme MinMax : jouer pour lui les actions qui maximisent ses gains et jouer pour l'adversaire les actions qui minimisent les siens.

4.1.1.4. Variantes de MCTS

La méthode présentée ici est celle ayant obtenu le plus grand retentissement au sein de la communauté et unanimement considérée comme la principale méthode de Monte Carlo pour les jeux. De nombreuses méthodes, inspirées de MCTS, ont été par la suite inventées. Le déroulement général de l'apprentissage, ou les connaissances apprises, sont sensiblement différentes, mais elles reposent toutes sur des simulations aléatoires réalisées à la volée afin de calculer des renforcements. Il ne nous semble pas nécessaire de détailler précisément leur fonctionnement étant donné qu'il ne s'agit pas du coeur de notre recherche. Nous pouvons toutefois citer parmi celles-ci Nested Monte Carlo[Caz09], Monte Carlo Beam Search[Caz12], Nested Roll-out Policy Adaptation[Ros11a].

4.1.2 Description détaillée des composantes de l'apprentissage

Le déroulement d'un épisode se découpe en quatre phases distinctes : *descent*, *growth*, *roll-out* et *update*. Dans cette section, nous revenons plus en détail sur chacune d'entre elles. Les trois premières sélectionnent les actions réalisées au cours de l'épisode. Le contexte d'application et les enjeux poursuivis par chacune d'entre elles diffèrent sensiblement. À chacune de ces phases est associée une politique dédiée que nous allons présenter sommairement. La dernière phase précise comment l'agent va apprendre de son expérience. Cette phase a une importante influence sur le comportement de l'agent. Nous en précisons les principaux enjeux.

4.1.2.1. Phase de *descent*

La phase de *descent* sélectionne les premières actions de l'épisode, pour les états actuellement couverts par l'arbre de recherche : c'est à dire de représentations brutes et précises. Cette phase dispose d'une politique dédiée $\pi_{descent}$ qui dépend des valeurs présentes dans les noeuds Q^{π}. Les valeurs Q^{π} sont ajustées lors de la phase *update*, à partir notamment des actions sélectionnées lors de la phase *descent*. Le rôle de la politique de *descent* est alors double. D'une part, elle sélectionne les meilleurs candidats aux vues des évaluations dont elle dispose. D'autre part, elle décide des prochains états à être renforcés lors de la prochaine phase *update*. Les deux phases *descent* et *update* sont étroitement liées et il n'est pas surprenant de voir des contributions portant sur les deux phases conjointement.

Verrou : Compromis exploration et exploitation À la différence des autres politiques, la politique de *descent* dépend des valeurs apprises. Les évaluations calculées à la volée ne sont pas fiables lors des premières visites car les moyennes sont alors calcu-

lées pour de trop petits échantillons (et sont donc sujettes à de fortes variations). Par ailleurs, la découverte de nouveaux états peut radicalement changer l'évaluation d'un état précédemment exploré (comme une défaite immédiate deux coups plus tard). Dans un tel contexte, le choix de l'action disposant de la meilleure évaluation moyenne peut s'avérer regrettable sur le long terme. À l'instar de toute politique d'apprentissage, la politique de *descent* doit assurer un bon compromis entre l'exploration d'états vraisemblablement moins bons et l'exploitation des meilleurs états de l'arbre (cf. Définition 3.18 à la page p. 54).

4.1.2.2. Phase de *growth*

La phase de *growth* intervient dès que les informations présentes dans l'arbre ne suffisent plus à décider de la prochaine action lors de la phase de *descent* : c'est à dire, lorsque les noeuds fils appartiennent à la frontière de l'arbre (les feuilles en théorie des graphes). La politique de *growth*, π_{growth}, détermine alors comment choisir le prochain état. Cet état sera celui ensuite ajouté à l'arbre de recherche.

Par extension la phase de *growth* définit les valeurs par défaut attribuées à chacun des états frontières qui n'ont pas encore été explorés depuis un état couvert par l'arbre. Dans l'algorithme canonique de MCTS, la phase de *growth* oblige à essayer exhaustivement l'ensemble des états atteignables avant d'approfondir la recherche d'un niveau supplémentaire.

Verrou : États non-couverts par l'arbre La politique π_{growth} se situe à la frontière entre des états pour lesquels l'agent dispose des informations apprises au cours de précédents épisodes et ceux pour lesquels il ne sait absolument rien. La profondeur atteinte par l'arbre de recherche dépendra du jugement porté par la politique de *growth* sur les états hors de l'arbre. Par défaut, la politique π_{growth} contraint d'avoir au moins une valeur pour chacun des états fils avant d'approfondir la recherche. C'est à dire que pour approfondir la recherche d'une action, un noeud feuille devra avoir été sélectionné autant de fois qu'il possède « d'états fils » au cours des épisodes suivants. Dans des jeux disposant d'un fort facteur de branchement comme au Go, une telle politique se heurte rapidement à la combinatoire du problème : le nombre de noeuds à explorer croît exponentiellement par rapport à la profondeur atteinte. Une politique de *growth* aménagée permet de restreindre ce phénomène.

4.1.2.3. Phase de *roll-out*

La phase de *roll-out* génère les actions restantes, une fois l'arbre de recherche quitté. La politique de *roll-out*, $\pi_{roll\text{-}out}$ ne dispose de plus aucune information pour déci-

der des prochaines actions. À défaut de pouvoir arbitrer les prochaines décisions, ces dernières sont réalisées aléatoirement [1]. La simulation d'actions aléatoires est caractéristique des méthodes de Monte Carlo. La génération aléatoire autorise, en principe, n'importe quelle évolution possible de la partie et offre ainsi une grande diversité dans les parties produites. La versatilité des simulations est généralement compensée par leur rapide génération. Lorsque les parties sont susceptibles d'être longues (comme pour le jeu de Go en 19x19), la majeure partie des actions de l'épisode sont sélectionnées au cours du *roll-out*. La vitesse à laquelle sont sélectionnées ces actions devient alors déterminante dans un contexte compétitif.

Verrou : simulations partiellement guidées Les simulations purement aléatoires garantissent la convergence des estimations vers leur valeur exacte car elle garantit suffisamment de diversité dans les actions sélectionnées. Or une politique purement aléatoire génère parfois des coups suffisamment aberrants pour ne pas être représentatifs du niveau d'un joueur débutant. L'ajout de connaissances élémentaires permettrait d'éviter ces nombreux écueils et d'accélérer ainsi l'apprentissage en produisant des simulations plus vraisemblables. Il s'agit en définitive de guider la politique de *roll-out* à partir de représentations supplémentaires. Une telle approche implique de considérer des représentations moins précises que celles naturellement utilisées dans l'arbre, mais aussi de veiller à la diversité des simulations produites.

4.1.2.4. Phase de *update*

La phase *update* renforce les actions sélectionnées au cours de l'épisode à partir du résultat de la simulation. Cette phase couvre deux aspects fondamentaux de l'apprentissage : le calcul de la récompense et comment l'agent adapte sa représentation en conséquence.

La phase *update* démarre dès qu'un état final a été atteint. Les règles du jeu permettent d'établir sans aucune ambiguïté le vainqueur de la partie simulée. La récompense est définie suivant l'issue de la partie. Par défaut, l'agent reçoit une récompense égale à 1 s'il atteint son objectif et sinon une récompense égale à 0. Les actions d'un même épisode sont générées suivant trois politiques différentes : $\pi_{descent}$, π_{growth} et $\pi_{roll\text{-}out}$. Par défaut, la récompense vient renforcer seulement les états de l'arbre ayant été traversés au cours de l'épisode : c'est à dire les états sélectionnés par la politique $\pi_{descent}$ et le nouvel état ajouté par la politique π_{growth}. L'ajustement est le même que

1. Dans des jeux comme le Go, la simulation n'est jamais purement aléatoire car sinon elle ne se terminerait jamais du fait des captures. Il faut en effet empêcher de jouer certains coups pour éviter cela comme par exemple « remplir ses propres yeux »

celui d'une méthode de Monte Carlo classique avec un taux d'amortissement $\gamma = 1$ (et r_T, la récompense finale) (cf. l'équation3.1) :

$$Q^{\pi}_{k+1}(s,a) = Q^{\pi}_{k}(s,a) + \underbrace{\frac{1}{k+1}}_{\alpha_k}\left[\underbrace{r_T - Q^{\pi}_{k}(s,a)}_{\delta}\right] \tag{4.1}$$

Verrou : choix de la récompense calculée Chaque épisode est une expérience à part entière, allant de la position courante du plateau jusqu'à un état final possible. Cette même expérience comporte de nombreuses informations qui sont autant de sources d'apprentissage. Par exemple la disposition des pièces, ou bien l'ordre suivant lequel ont été jouées certaines actions servent à produire d'autres renforcements complémentaires. La récompense calculée par défaut représente celle nécessaire a minima pour atteindre l'objectif initialement fixé. Cependant le calcul d'une récompense différente, ou bien de récompenses complémentaires peut aider à l'atteindre plus vite. L'enjeu consiste à identifier les observations pertinentes pour l'apprentissage de l'agent, au risque de ralentir l'exécution de la méthode.

Verrou : choix des représentations renforcées Le choix des représentations renforcées lors de la phase *update* est déterminant dans le processus d'apprentissage. Les évaluations apprises dépendent des seules récompenses accumulées au cours des épisodes. Par défaut, les seuls état renforcés correspondent aux états précis traversés au cours de l'épisode, à l'instar des méthodes canoniques d'apprentissage par renforcement. Cependant une même récompense peut servir à renforcer d'autres états semblables dans l'arbre, voire des représentations complètement différentes. Les représentations renforcées à la suite d'un épisode décideront de la signification des évaluations apprises.

4.1.3 Dynamique d'apprentissage de MCTS

MCTS appartient à la famille des méthodes d'apprentissage par renforcement, à l'instar d'une méthode de Monte Carlo. En tant que tel, sa dynamique d'apprentissage repose sur les interactions successives entre une politique et des valeurs d'état-actions : l'Itération de la Politique Généralisée (cf Définition 3.17 en page p. 52). Les valeurs d'état-actions sont ajustées par l'intermédiaire de la politique au cours d'une phase « d'évaluation de la politique ». À l'inverse, la politique est ajustée à partir des valeurs d'état-actions au cours d'une phase « d'amélioration de la politique ». L'IPG ne décrit pas le déroulement exact de l'algorithme mais met en avant les interactions entre représentation et processus de résolution. Dans le cadre de l'apprentissage par renforcement classique, il s'agit systématiquement d'une interdépendance.

En l'occurrence, la dynamique d'apprentissage de MCTS est légèrement plus complexe. En effet, ce n'est pas une seule politique qui décide des actions jouées, mais les politiques des trois premières phases. Ces trois politiques décident de l'issue de l'épisode et par conséquent du renforcement associé lors de la phase *update*. À l'inverse, la politique de *descent* est la seule à bénéficier en retour des valeurs présentes dans l'arbre de recherche afin d'ajuster ses propres choix. L'IPG de MCTS est résumée dans la figure 4.2. Les différentes phases de MCTS correspondent, dans une certaine mesure, aux phases d'amélioration et d'évaluation de la politique : les phases de *descent*, *growth* et *roll-out* pour l'amélioration de la politique et la phase *update* pour l'évaluation de la politique. L'unique interdépendance est entre la politique de *descent* et les valeurs présentes dans l'arbre. Nous nous reposerons sur ce schéma pour décrire les différentes dynamiques d'apprentissage à l'oeuvre dans les différentes améliorations de MCTS.

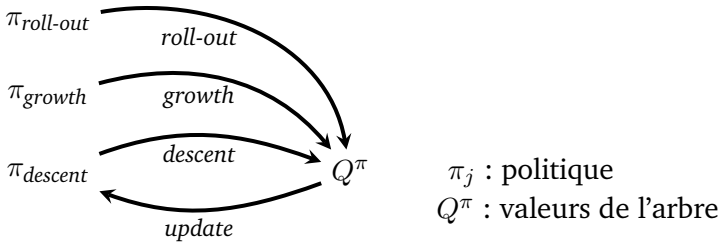


Figure 4.2.– Itération de la Politique Généralisée de MCTS

4.2 État de l'art des améliorations de MCTS

La méthode MCTS présentée n'a jamais été appliquée comme telle. Elle a systématiquement été complétée d'améliorations, mêmes infimes, lors de sa mise en application. Dans cette section nous recensons les améliorations apportées à MCTS dans la littérature. Cette méthode a donné lieu à de très nombreuses contributions. L'état de l'art proposé n'est pas exhaustif de l'ensemble des contributions. Pour une présentation plus exhaustive des différentes avancées, nous invitons le lecteur à se référer à l'inventaire proposé par Browne et al.[Bro+12].

Dans le présent travail, nous nous intéressons en particulier à la dynamique d'apprentissage de MCTS. Nous avons choisi de présenter, dans un premier temps, les principales améliorations apportées à MCTS qui ne remettent pas en cause les interactions de cette dynamique dans la section 4.2.1. Ensuite nous présenterons les améliorations de MCTS introduisant une nouvelle dynamique d'apprentissage dans la section 4.2.2. Un tableau récapitulatif des améliorations présentées est consultable en section 4.2.3.

4.2.1 Améliorations neutres auprès de la dynamique d'apprentissage

Les améliorations présentées dans cette section modifient localement chacune des quatre phases présentées, en réponse aux verrous exposés dans la section 4.1.2. Si les différentes phases sont individuellement modifiées, les interactions entre politiques et valeurs d'action restent inchangées : aucune de ces améliorations ne remet en cause la dynamique d'apprentissage présentée en figure 4.2.

Suivant notre perspective de travail, ces améliorations ont un moins grand impact sur l'évolution de l'apprentissage de l'agent. Il n'en reste pas moins que les principales améliorations employées actuellement dans les programmes de Go sont présentées dans cette section et ne modifient donc pas fondamentalement la dynamique d'apprentissage.

4.2.1.1. Phase de *descent*

La phase de *descent* couvre les actions sélectionnées à partir des valeurs présentes dans l'arbre. Comme nous l'avons évoqué dans la section 4.1.2.1, la politique de *descent* doit identifier les actions les plus prometteuses malgré des évaluations moyennes imprécises. Les politiques de *descent* améliorées utilisent d'autres évaluations supplémentaires afin de cerner rapidement les actions les plus prometteuses. Nous présentons ici les améliorations suivantes :

— *Upper Confidence applied to Trees* (UCT) ;

— *Rapid Action Value Estimate* (RAVE) ;

— *Criticality* ;

— *Group Node*.

Les deux premières sont celles habituellement utilisées dans les programmes de Go. Les deux suivantes apportent un moindre gain au programme, mais il nous semble intéressant de les mentionner pour leur originalité.

Upper Confidence applied to Trees (UCT) L'évaluation UCT établit formellement un compromis entre exploration et exploitation (cf. Définition 3.18 à la page p. 54), pour chaque noeud de l'arbre[KS06]. La mise en situation classique pour traiter ce compromis dans la littérature est le problème de bandit à $n-$bras (cf. Annexe A.2). L'évaluation UCT applique pour chaque prise de décision une évaluation employée initialement dans les problèmes de bandit à $n-$bras : l'évaluation *Upper Confidence Bound* (UCB). L'évaluation par défaut est alors complétée par une autre évaluation dépendant du nombre de fois que l'action a été choisie comme présentée dans

l'équation (4.2). Cette valeur complémentaire permet d'équilibrer les valeurs des actions ayant été souvent sélectionnées, et donc fiables, de celles dont la valeur moyenne est encore susceptible d'évoluer. L'évaluation UCT dépend d'un paramètre c [2] qui régule le compromis entre exploration et exploitation : plus c sera élevé, plus la politique privilégiera les actions exploratoires.

$$Q^{\pi}_{\text{UCT}}(s,a) = Q^{\pi}(s,a) + c\sqrt{\frac{\ln n_s}{n_{(s,a)}}} \quad , \tag{4.2}$$

tel que

$Q^{\pi}(s,a)$	: évaluation de l'action a depuis l'état s
c	: coefficient d'exploration
n_s	: nombre de renforcements pour l'état s
$n_{(s,a)}$	: nombre de renforcements pour l'action a depuis l'état s

Rapid Action Value Estimate (RAVE) Les évaluations RAVE [GS11] sont des évaluations d'action légèrement biaisées mais calculées à partir d'un plus grand nombre de renforcements que celles par défaut. Ce faisant, les évaluations RAVE éludent le compromis entre exploration et exploitation en comparant les actions à partir d'évaluations stables plus rapidement. L'évaluation RAVE appartient à la famille des évaluations *All-Move-As-First* (AMAF) [HP09], c'est à dire des évaluations ne tenant pas compte de l'ordre suivant lequel sont joués les actions au cours de l'épisode. L'amélioration RAVE (comme toutes les évaluations AMAF) est étroitement liée à la façon dont les actions sont renforcées lors de la phase *update* (nous reviendrons sur cet aspect dans la section 4.2.1.4). De telles évaluations fournissent rapidement de bonnes estimations de la valeur d'un état, mais le biais introduit est susceptible de mal orienter la politique de *descent* dans certaines situations [TM11]. C'est pourquoi cette évaluation est parfois utilisée seulement lors des premiers renforcements d'un noeud : l'évaluation par défaut, ou une autre évaluation sans biais comme UCT prend alors progressivement le relais [Sil09].

Criticality La *criticality* [Cou09 ; Pel+09] propose une évaluation complémentaire afin d'identifier rapidement les actions statistiquement décisives de l'issue de la partie. À l'instar de RAVE, il s'agit d'éluder le compromis entre exploration et exploitation à la différence près que la *criticality* repose sur une évaluation complémentaire (calculée indépendamment) et non sur une valeur moyenne biaisée. Cette évaluation

2. L'évaluation UCB ne comporte pas ce paramètre c car le problème de bandit à $n-$bras se limite en principe à des épisodes ne comprenant qu'une seule décision à prendre. Dans le cas de décisions successives, comme pour UCT, il faut tenir compte qu'en explorant de nouveaux états l'agent bénéficie d'informations supplémentaires qui seront susceptibles d'influer sur le jugement [KS06].

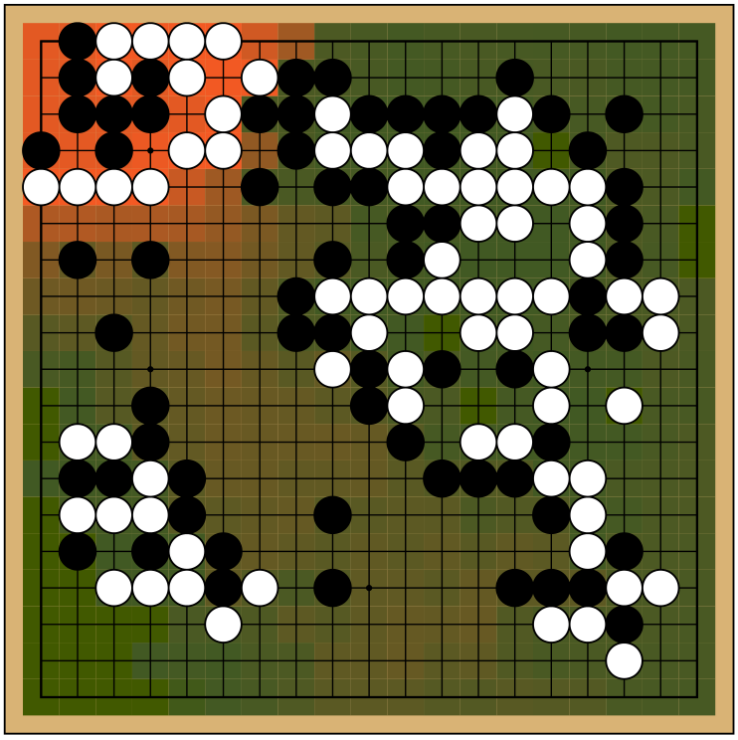


Figure 4.3.– Illustration de la *criticality* pour une position de Go [3] : l'évaluation de *Criticality* obtenue pour chacune des intersections permet d'identifier les zones décisives de l'issue de la partie (sur la figure en haut à gauche) et ainsi servir à orienter la recherche.

a été initialement proposée pour le jeu de Go mais elle est en principe extensible à tout autre jeu de pose (comme Hex ou Havannah). Pour chaque intersection, la *criticality* évalue la corrélation entre la possession de cette intersection en fin de partie et l'issue de la partie pour le joueur qui la contrôle (victoire ou défaite). La formule du calcul est détaillée dans l'équation (4.3). Cette évaluation nécessite de calculer des récompenses supplémentaires à la fin de chaque épisode (nous reviendrons sur cet aspect dans la section 4.2.1.4). La valeur est comprise entre -1 et 1 [4]. Une forte valeur (positive ou négative) suggère que le contrôle de cette intersection est corrélée avec l'issue de la partie. À l'inverse une *criticality* proche de zéro signifie que le contrôle de l'intersection est statistiquement indépendant d'une victoire ou d'une défaite. En pratique, la *criticality* vient s'ajouter à l'évaluation de l'action correspondant à cette intersection [BG12 ; Pel+09]. Pellegrino et Drake[Pel+09] ont montré que cette évaluation donne une approximation encore plus rapide que RAVE pour les larges

3. Cette image est extraite du diaporama [Cou09] et est reproduite ici avec l'autorisation de l'auteur.
4. La *criticality* correspond au calcul d'une covariance entre deux variables aléatoires[Pel+09] : le contrôle de l'intersection et la victoire. En tant que tel, sa valeur de covariance est comprise entre -1 et 1.

espaces de recherche mais qui est dépassée en terme de performance par RAVE puis UCT, à mesure que le nombre de simulations augmente.

$$\text{Crit}(a) = Q_{win}(a) - (Q_B(a).Q_B + Q_W(a).Q_W) \quad , \tag{4.3}$$

tel que

$Q_{win}(a)$: moyenne de gagner en possédant a
$Q_B(a), Q_W(a)$: moyenne de posséder a pour le joueur noir, respectivement blanc
Q_B, Q_W : moyenne de gagner pour le joueur noir, respectivement blanc

Group Node Les *Group Node*[Lor08] introduisent des niveaux intermédiaires dans l'arbre de recherche afin de réduire artificiellement le facteur de branchement de l'arbre. Cette amélioration ne traite pas directement du compromis entre exploration et exploitation mais permet d'en faciliter la résolution. En effet, les actions les plus prometteuses sont identifiables plus rapidement lorsqu'elles sont en compétition avec un plus petit nombre d'actions. En outre, l'apprentissage de l'agent est accéléré [VM12] à nombre constant d'épisodes. Les *Group Node* regroupent un ensemble d'actions qui sont évaluées conjointement comme n'importe quelle autre action de l'arbre. La sélection de la prochaine action se déroule alors en deux temps lors de la phase de *descent*. Dans un premier temps la politique $\pi_{descent}$ sélectionne parmi les *Groupe Node* le meilleur groupe d'action. La politique $\pi_{descent}$ sélectionne l'action à jouer, parmi les actions de ce groupe. À la fin de l'épisode, les *Group Node* sont renforcés comme n'importe quel autre noeud de l'arbre de recherche. La principale difficulté consiste à définir comment construire les groupes [VM12 ; CBK08]. La découpe semble naturelle dans des jeux où le tour de jeu se déroule en deux étapes distinctes, comme Amazons par exemple [Lor08]. Dans des jeux comme le Go, cette découpe repose sur des connaissances expertes [Sai+07].

4.2.1.2. Phase de *growth*

La phase de *growth* se situe à la frontière entre les actions présentes dans l'arbre et celles en dehors. Comme nous l'avons évoqué dans la section 4.1.2.2, la profondeur de la recherche dépend de comment la politique π_{growth} considère les états non-explorés. Les améliorations présentées attribuent des valeurs aux états non-explorés afin de pouvoir y appliquer une politique de sélection proche de la politique $\pi_{descent}$ voire identique. Nous présentons seulement les deux qui nous paraissent les plus emblématiques :

— *First Play Urgency* (FPU) ;

— *Prior knowledge*.

La première amélioration est générique à n'importe quel jeu mais n'offre qu'un contrôle limité sur l'exploration des états hors de l'arbre. La seconde établit une distinction plus fine parmi les états hors de l'arbre mais nécessite l'ajout de connaissances spécifiques au jeu. À la différence de la politique de *descent*, les politiques présentées ici n'évoluent pas suivant les résultats des épisodes : elles s'appliquent de la même manière tout au long de la partie.

First Play Urgency (FPU) La *First Play Urgency*[WG07] (FPU) définit la valeur par défaut attribuée à tous les états hors de l'arbre. Au cours de la phase de *growth*, la politique classique de $\pi_{descent}$ est ensuite appliquée telle que les états déjà explorés bénéficient de leur évaluation actuelle (UCT,RAVE, etc) et ceux non-explorés ont la FPU comme valeur. La valeur de FPU établit un compromis entre exploration et exploitation au niveau des états non-explorés. Pour de faibles valeurs de FPU, la politique π_{growth} privilégiera l'exploitation des états déjà couverts. À l'inverse, la politique favorisera l'exploration de nouveaux états pour de fortes valeurs de FPU. À titre de comparaison, la politique de *growth* par défaut revient à attribuer une FPU infinie aux états non-explorés. La FPU représente un premier ajustement possible de la politique de *growth*. Néanmoins elle correspond à la représentation la plus floue qui soit : elle est appliquée uniformément pour tous les états non-explorés. Un plus grand discernement nécessite de bénéficier d'informations supplémentaires comme présenté ci-dessous.

Prior knowledge Les *Prior knowlegde* [5] fournissent une première évaluation aux états non-explorés à partir de représentations complémentaires. Une fois la phase de *descent* terminée, la politique de *growth* se contente d'initialiser tous les états fils du dernier état couvert et applique ensuite une dernière fois la même politique que celle de la phase de *descent*. Les représentations mobilisées pour calculer ces évaluations sont généralement des représentations floues voire enrichies comme des schémas spatiaux, experts [Enz+09 ; GS08 ; Cha+08] ou des réseaux de Bradley Terry[Cou07a]. Les valeurs attribuées sont établies au préalable suivant un apprentissage supervisé[Cou07a] ou un apprentissage par renforcement indépendant [GS08] : la politique d'initialisation reste la même tout au long de la partie, bien que les initialisations soient différentes d'une action à l'autre. Ces premières approximations permettent d'amorcer la recherche autour des actions considérées comme prometteuses *a priori*. Les renforcements obtenus lors des épisodes consécutifs viendront l'ajuster le cas échéant.

5. Le terme *Prior knowledge* est inspiré de celui proposé par le programme Fuego[Enz+09]. Il recouvre ici de nombreuses heuristiques différentes reposant sur le même principe [Cha+08 ; Cou07a]

4.2.1.3. Phase de *roll-out*

La phase de *roll-out* génère aléatoirement les dernières actions de l'épisode. Comme nous l'avons évoqué dans la section 4.1.2.3, les simulations aléatoires gagnent à bénéficier de connaissances élémentaires afin de rendre les simulations vraisemblables. Les améliorations présentées ici contraignent la politique de *roll-out* afin d'intégrer des rudiments de tactiques. Les représentations mobilisées sont alors spécifiques au jeu concerné. À l'instar de la politique de *roll-out* par défaut, les politiques présentées ici n'évoluent pas suivant les résultats des épisodes : elles s'appliquent de la même manière tout au long de la partie.

Politique *Sequence-like* Les politiques de *roll-out* « *Sequence-like*[6] » mobilisent des représentations complémentaires afin de rendre les simulations produites plus vraisemblables. Initialement de telles politiques utilisaient essentiellement des schémas spatiaux[Gel+06] pour guider les simulations. Depuis d'autres représentations ont été utilisées comme les schémas experts[Enz+09] ou les réseaux de Bradley-Terry[Cou07a]. Cependant, les actions spatialement proches de la dernière action sont systématiquement privilégiées : c'est notamment grâce à cette contrainte que les actions générées semblent se répondre localement. Les politiques *Sequence-like* ont considérablement amélioré les performances des programmes de Go. Néanmoins,les raisons de leur succès restent encore difficiles à identifier[ST09].

4.2.1.4. Phase de *update*

La phase *update* renforce les évaluations de l'arbre à partir du résultat de la simulation. Comme nous l'avons évoqué dans la section 4.1.2.4, cette phase remplit un double rôle : elle définit les récompenses perçues et détermine les représentations qui seront renforcées. Les améliorations proposées dans la littérature jouent sur les deux rôles de cette même phase. Nous présentons ici les améliorations suivantes :

— *Rapid Action Value Estimate* (RAVE) ;

— *Criticality* ;

Les deux améliorations présentées font naturellement écho à celles présentées pour la phase *descent*. Cependant nous les présentons ici du point de vue de la phase *update* : elles permettent notamment de mettre en lumière chacun des verrous identifiés pour cette phase. La mise à jour de RAVE renforce un plus grand nombre de représentation que la mise à jour par défaut, tandis que la mise à jour *Criticality* considère d'autres récompenses que celles habituellement perçues.

6. La politique *Sequence-like* a été initialement introduite par le programme Mogo[WG07]. De nombreux programmes de Go s'en inspirent directement pour concevoir leur propre politique de *roll-out*[Enz+09 ; BG12 ; Shi11]. Nous utilisons ce terme pour couvrir l'ensemble des *roll-out* experts inspirés cette politique-ci.

***Rapid Action Value Estimate* (RAVE)** La mise à jour RAVE partage le renforcement à un plus grand nombre d'états, afin de produire des évaluations fiables plus rapidement. Comme les arbres sont une représentation précise, les évaluations stockées d'une branche à l'autre de l'arbre de recherche sont sujettes à une forme de redondance (cf. Définition 3.9). Si la représentation même de l'arbre de recherche n'autorise pas de mutualiser des informations semblables, RAVE distribue le même renforcement à des états jugés comme semblables à ceux renforcés par défaut. Par défaut, l'évaluation d'un couple état-action est renforcé à chaque fois que l'action a été jouée immédiatement après cet état. Dans la mise à jour RAVE, un couple état-action est aussi renforcé lorsque l'action a été jouée bien plus tard au cours de l'épisode et ceci même si cette action a été sélectionnée avec une autre politique (comme celle du *roll-out* par exemple). Ainsi des états n'ayant pas été traversés au cours de l'épisode bénéficient de ce renforcement supplémentaire qui stabilise leur évaluation. RAVE autorise dans une certaine mesure un partage de l'information sur les différentes branches de l'arbre[GS11]. Cette généralisation reste néanmoins limitée par la représentation utilisée pour stocker les évaluations : l'arbre de recherche.

Criticality La *criticality* observe d'autres éléments de l'épisode, comme la disposition finale du plateau, afin de calculer des évaluations complémentaires de celles par défaut. Pour chaque intersection du goban, la *criticality* requiert au moins deux évaluations en plus de celles par défaut : la probabilité de posséder l'intersection pour un joueur[7] ainsi que la probabilité de gagner pour le joueur possédant cette intersection. Ces deux probabilités sont estimées à partir de renforcements successifs au même titre que l'évaluation par défaut. Depuis l'état final deux récompenses supplémentaires sont calculées par intersection. Pour la première évaluation, une récompense de 1 est perçue si l'intersection est possédée à la fin par un des deux joueurs et sinon 0. Pour la seconde évaluation, une récompense de 1 est perçue lorsque l'intersection est de la couleur du vainqueur et sinon 0. Les récompenses sont ensuite rétro-propagées dans l'arbre au cours de la phase *update*, de la même manière que la récompense par défaut.

4.2.2 Améliorations avec une influence sur la dynamique d'apprentissage

Par défaut, les phases de *growth* et *roll-out* ne dépendent pas de valeurs apprises à la volée. Ces deux politiques n'apprennent pas des récompenses perçues : cela s'illustre dans l'IPG (cf. figure 4.2) par une absence de flèche entrante pour aucune des deux. Dans cette section, nous présentons des politiques de *growth* et *roll-out* qui évoluent au cours des épisodes.

7. La probabilité pour l'autre joueur peut être déduite en calculant la probabilité complémentaire.

À l'image de la politique de *descent*, une politique évolutive requiert une représentation adéquate pour accumuler les renforcements successifs. L'arbre de recherche se prête parfaitement à la politique de *descent* mais ne convient pas pour autant aux politiques concernées ici. Dans un premier temps, nous traitons des améliorations qui suggèrent justement de réutiliser des connaissances présentes dans l'arbre de recherche pour influencer les politiques de *growth* et *roll-out*. Comme l'arbre de recherche est lui-même évolutif, ces politiques le seront aussi en conséquence. Ensuite, nous présentons les améliorations qui utilisent des représentations indépendantes de l'arbre de recherche. Ces améliorations traitent exclusivement de la phase de *roll-out* et mobilisent comme représentation des schémas partiels (cf. section 3.1.2.2). Enfin nous évoquons les principales modifications des améliorations abordées sur la phase *update*.

4.2.2.1. Phase de *growth* influencée par les valeurs de l'arbre

La politique de *growth* traite des actions situées à la frontière entre celles présentes dans l'arbre et celles en dehors. Comme nous l'avons évoqué en section 4.1.2.2, cette politique a une influence directe sur la profondeur de la recherche. Il s'agit d'un choix difficile à réaliser. L'intérêt de réutiliser des valeurs déjà présentes dans l'arbre consiste à reprendre les évaluations déjà accumulées pour des états semblables afin d'initialiser la valeurs de ces états. Cette heuristique est depuis longtemps utilisée par les programmes de jeux sous le nom d'*history heuristic*. La dynamique d'apprentissage induite par cette heuristique est résumée par la figure 4.4.

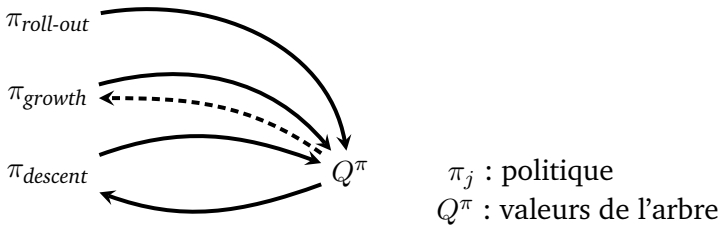


Figure 4.4.– IPG de MCTS avec une *history heuristic*

History heuristic L'*History heuristic* a été initialement proposée par Schaeffer[Sch89] pour le processus de résolution $\alpha - \beta$. À l'instar de la mise à jour de RAVE, il est alors supposé que la valeur d'un couple état-action reste inchangée si cette même action est jouée plus tard au cours de l'épisode. Les valeurs calculées pour des actions dans les premiers niveaux de l'arbre de recherche sont alors réutilisées pour ces mêmes actions à des niveaux plus profonds. Cette heuristique a été adaptée pour MCTS sous le nom de *grand-father heuristic*[GS07]. La valeur de l'action est alors reprise à deux niveaux de moins exactement. Si l'influence de cette heuristique sur les performances de MCTS reste limitée, la démarche n'en reste pas moins intéressante. Déjà dans son

article de 1989, Schaeffer présentait son heuristique comme un pas en avant vers l'autonomie du processus de résolution[Sch89] : l'agent construit de lui-même ses connaissances.

4.2.2.2. Phase de *roll-out* influencée par l'arbre de recherche

La phase de *roll-out* génère les dernières actions de l'épisode suivant une politique purement aléatoire par défaut. Cependant cette politique gagne à être complétée de connaissances élémentaires afin de rendre les simulations vraisemblables, comme nous l'avions évoqué en section 4.1.2.3. Les améliorations présentées dans cette section suggèrent de réutiliser des connaissances déjà présentes dans l'arbre, à l'instar de l'*history heuristic*. La dynamique d'apprentissage est alors modifiée en conséquence, comme cela est résumé dans la figure 4.5.

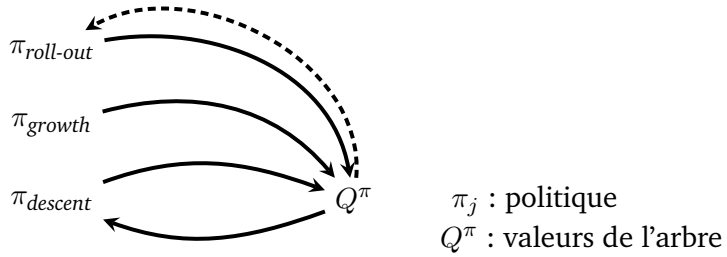


Figure 4.5.– IPG de MCTS avec une politique $\pi_{roll\text{-}out}$ influencée par l'arbre de recherche

La première amélioration proposée, Pool-RAVE repose sur les valeurs directement présentes dans l'arbre de recherche. La *Roll-out history heuristic* nécessite de construire une représentation complémentaire directement influencée par les valeurs dans l'arbre.

Pool RAVE La politique Pool-RAVE [RTT11] guide la politique de *roll-out* à partir des valeurs RAVE présentes dans l'arbre de recherche (cf. section 4.2.1.1). Au cours de la phase de *descent*, la politique mémorise les actions disposant d'une forte valeur RAVE, mais qui n'ont pas été pour autant sélectionnées. Lors de la phase de *roll-out*, les actions ainsi accumulées sont générées suivant une probabilité fixée au préalable. Dans une certaine mesure, la politique de *roll-out* Pool-RAVE catalyse l'apprentissage des évaluations RAVE dans l'arbre de recherche. En effet lorsqu'une action sera sélectionnée par Pool-RAVE, elle sera irrémédiablement renforcée ensuite par la mise à jour RAVE (cf. section 4.2.1.4). Les évaluations RAVE seront d'autant plus vite évaluées. C'est notamment cette raison qui nous semble expliquer les succès de la méthode dans les jeux de Go et Havannah.

***Roll-out* history heuristic** Les *Roll-out history heuristics* [8] [DU07] orientent la simulation à partir de connaissances présentes dans l'arbre de recherche, mais portées par une autre représentation. Lors de la phase de *roll-out*, la politique privilégie les actions ayant été le plus sélectionnées sur l'ensemble de l'arbre. La *first order heuristic* considère l'action seule et indépendamment de tout contexte : schéma temporel de taille 1. La *second order heuristic* considère les actions les plus fréquentes en réponse à une action de l'adversaire : schéma temporel de taille 2. Cette amélioration en recouvre d'autres, comme la TreeOnly-MAST [FB10], qui considèrent la valeur moyenne (et non la fréquence) d'une action seule et indépendamment de tout contexte. Les représentation mobilisées ici sont limitées à une ou deux actions consécutives. Nous allons par la suite proposer d'étendre cette représentation dans notre première contribution.

4.2.2.3. Phase de *roll-out* influencée par des schémas indépendants

La politique de *descent* bénéficie des évaluations présentes dans l'arbre de recherche pour orienter la zone de l'espace de recherche à explorer. La représentation et la politique s'ajustent mutuellement au travers d'une interdépendance fondamentale à toute IPG (cf Définition 3.17 à la page 52). À l'image de la politique de *descent*, les améliorations proposées dans cette section introduisent une nouvelle interdépendance entre la politique de *roll-out* et une nouvelle représentation. Cette représentation sera aussi renforcée par les résultats des simulations mais ne concerne que les actions produites par la politique de *roll-out*, comme cela est résumé dans la figure 4.6.

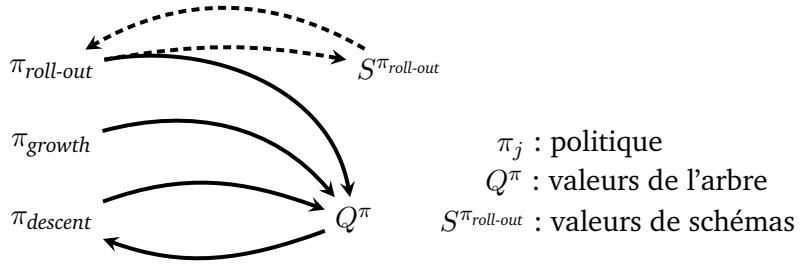


Figure 4.6.– IPG de MCTS avec une politique $\pi_{roll\text{-}out}$ influencée par une représentation indépendante de l'arbre

Par définition, les représentations introduites pour la politique de *roll-out* ne peuvent pas être aussi précises que dans un arbre de recherche. Il s'agit pour l'essentiel de schémas partiels pour un contexte très restreint (très flou). Les trois premières améliorations proposées *N-gram Selection Technique*, *Last Good Reply Forgetting* et

8. Les auteurs avaient proposé initialement le nom de *history heuristic* en inspiration de l'heuristique de Schaeffer [Sch89]. Pour ne pas confondre avec celle appliquée pour la phase de *growth*, nous avons choisi d'y ajouter le nom de la phase.

Contextual Monte Carlo utilisent des schémas temporels (ou des représentations proches). Elles sont en principe génériques à n'importe quel jeu. La Technique de sélection de N-grammes spatiaux est la seule, à ce jour, à avoir obtenu un gain de performances pour des schémas spatiaux malgré les limitations pointées par Lew dans son mémoire de doctorat [Lew11]. Elle a été appliquée pour le jeu de Go exclusivement.

N-gram Selection Technique (NST) NST[TWB12] modifie la politique de *roll-out* à partir de schémas temporels valués d'une taille inférieure ou égale à 3 (cf. Définition 3.10 à la page p. 37). À chaque étape de la simulation de *roll-out*, les seuls schémas temporels en compétition sont ceux dont le contexte coïncide avec les dernières actions simulées. La réponse sélectionnée est tirée aléatoirement suivant les valeurs associées à chacun des schémas. Les schémas temporels ont l'avantage d'être applicables à n'importe quel jeu. À cet effet, ils ont été notamment appliqués au *General Game Playing* (GGP). La NST englobe d'autres heuristiques développées par le passé comme la Move-Average-Sampling-Technique [FB10] qui traite de schémas temporels de taille 1.

Last Good Reply Forgetting (LGRF) LGRF[Dra09 ; BD10] modifie aussi la politique de *roll-out* à partir de schémas temporels d'une taille inférieure ou égale à 3 (cf. Définition 3.10 à la page p. 37). À la différence de méthodes comme NST, chaque contexte ne dispose que d'une seule et unique réponse. Lors de la phase *roll-out*, la réponse est sélectionnée dès que les actions du contexte coïncident aux dernières actions simulées. Cette amélioration a été initialement conçue pour le jeu de Go[Dra09] et a depuis été adaptée pour le General Game Playing [TWB12]. Cette méthode est proche de NST dans les représentations qu'elles mobilisent. Néanmoins la politique d'application et la mise à jour les distinguent très nettement.

Contextual Monte Carlo (CMC) CMC[RT10] mobilise des tuiles de taille 2 valuées pour orienter la politique de *roll-out*. À la différence des schémas temporels, une tuile ne considère pas un ordre particulier : les actions n'ont pas besoin d'être jouées consécutivement et suivant un ordre précis. Par conséquent à chaque étape de la simulation, les tuiles en compétition sont celles dont une des deux actions a déjà été simulée. La politique alors proposée sélectionne avec probabilité fixe celle avec la meilleure évaluation. CMC a été appliqué au jeu de Havannah et correspond dans cet état de l'art à la seule amélioration n'incluant aucune relation de proximité dans sa représentation : aussi bien spatiale que temporelle.

Technique de Sélection de N-grammes Spatiaux (TSNS) Les TSNS[Bas+14] [9] orientent la politique de *roll-out* à partir de N-grammes spatiaux d'une taille inférieure ou égale 2. Les N-grammes spatiaux sont équivalents à des schémas temporels où chacune des actions est remplacée par un schéma spatial. Cela signifie en l'occurrence que l'un des schémas spatiaux correspond au contexte et l'autre donne le schéma réponse. Un N-gramme spatial de taille 1 est équivalent à un schéma spatial classique valué. Au cours de la phase *roll-out*, les seuls N-grammes de taille 2 en compétition sont ceux dont le schéma « contexte » coïncide avec la disposition des pierres autour du dernier coup joué. Parmi les N-grammes avec la meilleure évaluation, un schéma spatial « réponse » est sélectionné, puis il est ensuite appliqué. Si aucun N-gramme de taille 2 n'est retenu, le meilleur schéma spatial légal (N-gramme de taille 1) est appliqué autour du dernier coup joué. Cette méthode a été proposée pour le jeu de Go.

4.2.2.4. Update

Les modifications apportées à la dynamique d'apprentissage introduisent de nouvelles représentations. À partir du moment où ces représentations dépendent des résultats des simulations, la phase *update* est nécessairement modifiée en conséquence ; et notamment suivant les verrous évoqués en section 4.1.2.4. En particulier, les représentations renforcées à la fin de chaque épisode comprennent les représentations introduites en plus de l'arbre de recherche. Par ailleurs, nous verrons que certaines de ces représentations requièrent une autre récompense que le seul résultat. Nous présentons ici, du point de vue de la phase *update*, les améliorations suivantes :

— *Roll-out history heuristic* ;

— *Contextual Monte Carlo* (CMC), *N-gram Selection Technique* (NST) et Technique de Sélection de N-grammes spatiaux (TSNS) ;

— *Last Good Reply Forgetting* (LGRF).

Les deux premières améliorations traitent essentiellement du choix des représentations renforcées au cours de la phase *update*. Nous avons regroupé les trois améliorations CMC, NST et TSNS du fait des similarités qu'elles entretiennent dans le renforcement de leur représentation. Si le choix des représentations renforcées par LGRF est semblable à celui des améliorations précédentes, le renforcement de LGRF est sensiblement différent. Nous évoquerons notamment les implications d'un tel choix.

9. Les auteurs de cette méthode n'ont pas proposé de nom spécifique pour décrire leur méthode. Nous avons choisi Technique de Sélection de N-grammes Spatiaux afin de mettre en relief la principale différence avec la N-gram Selection Technique.

Roll-out history heuristic La *roll-out history heuristic* propose d'extraire des informations présentes dans l'arbre de recherche sous une représentation simplifiée, en l'occurrence l'évaluation d'une action seule sans aucun autre contexte temporel. Une extraction à la suite de chaque phase *update* serait bien trop coûteux en terme de temps de calcul. La meilleure façon d'extraire cette information revient à renforcer cette représentation complémentaire conjointement à l'arbre de recherche. En d'autres termes, les actions renforcées dans la représentation complémentaire correspondent à celles qui ont été sélectionnées au cours de la phase *descent* de l'épisode. Il s'agit d'une subtilité sur laquelle nous allons nous reposer dans notre première contribution pour « extraire » de l'arbre de recherche des représentations plus complexes.

CMC, NST, TSNS Ces trois améliorations proposent d'établir une interdépendance entre la politique de *roll-out* et les nouvelles représentations introduites. À l'image des états couverts par l'arbre de recherche, chacune des nouvelles représentations sera renforcée lorsqu'elle sera apparue au cours de la phase de *roll-out*. Par exemple, un N-grammes de l'amélioration NST sera renforcé dès qu'une séquence d'action au cours de l'épisode coïncidera avec la totalité de la représentation (contexte et réponse)[TWB12]. Il en va de même pour les N-grammes spatiaux de la méthodes TSNS[Bas+14] et les couples d'actions de CMC[RT10]. À l'instar des états de l'arbre, la récompense perçue est dépendante de l'issue de la partie. Ces représentations seront renforcées de la même manière que n'importe quelle méthode de Monte Carlo : par un calcul incrémental de la moyenne (cf. Equation 4.1).

LGRF Les schémas temporels de LGRF seront renforcés à partir des actions qui ont été sélectionnées au cours de la phase de *roll-out* comme les améliorations précédentes. À la différence de NST, les schémas temporels de LGRF ne comportent qu'une seule et unique réponse pour un même contexte temporel. Par conséquent, le renforcement ne portera pas sur l'ajustement d'une évaluation mais changera directement la réponse associée à ce contexte. Lors de la phase *update*, les réponses apportées par le vainqueur de la partie sont attribuées à chacun des contextes apparaissant au cours de l'épisode. À l'inverse, si le perdant de la partie a joué la réponse d'un schéma temporel actuellement mémorisé par LGRF, celle-ci est oubliée.

Ce renforcement se distingue très nettement de ceux des améliorations ci-dessus du fait de sa « volatilité ». En effet une valeur moyenne sert à évaluer de manière absolue une représentation. À mesure que la représentation est renforcée, l'évaluation se stabilise progressivement vers une estimation de cette représentation. À tel point que la réponse à un même contexte parmi tous les schémas temporels couverts par une amélioration comme NST tend à rester la même au bout d'un certain temps (ou bien évolue très lentement). À l'inverse dans LGRF, la réponse associée à un même

contexte est susceptible de changer complètement après n'importe quel épisode. Un tel renforcement permet de suivre au plus près les changements brutaux de situations et de s'adapter au mieux à l'évolution de la partie. Nous reviendrons sur un renforcement aussi volatile dans le prochain chapitre pour traiter de l'une des variantes de notre première contribution.

4.2.3 Récapitulatif des améliorations

La table 4.1 détaille les améliorations pour MCTS suivant les différentes dynamiques d'apprentissage abordées. Pour chaque amélioration, nous précisons les représentations généralement utilisées pour les mettre en oeuvre.

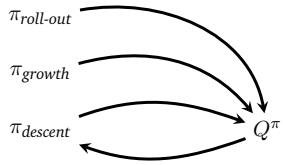
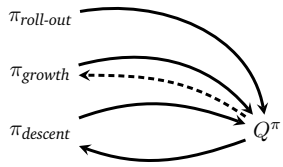
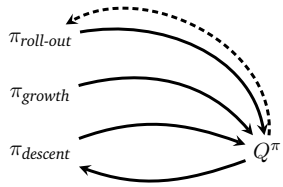
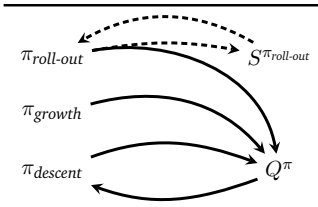
Dynamique d'apprentissage	Politique	Améliorations	Représentation support
	$\pi_{descent}$	*UCT*, *RAVE*, *Criticality*, *Group Node*	Arbre de recherche
	π_{growth}	*FPU*, *Prior Knowledge*	Schémas partiels, Réseaux Bradley-Terry
	$\pi_{roll\text{-}out}$	*Sequence-like*	Schémas partiels, Réseaux Bradley-Terry
	π_{growth}	*History Heuristic*	Arbre de recherche
	$\pi_{roll\text{-}out}$	*Pool-RAVE*, *Roll-out history heuristic*	Arbre de recherche, Actions seules
	$\pi_{roll\text{-}out}$	*NST*, *LGRF*, *CMC*, *TSNS*	Schémas temporels, Schémas spatiaux

Table 4.1.– Récapitulatif des améliorations de MCTS pour le domaine des jeux

La majorité des contributions pour MCTS n'affecte pas fondamentalement la dynamique d'apprentissage par défaut. Les améliorations présentées pour cette même

dynamique d'apprentissage sont celles habituellement utilisées par les programmes compétitifs. À titre d'exemple le programme Fuego[Enz+09] sur lequel nous allons ensuite travailler dispose des améliorations UCT, RAVE,*Prior Knowledge* et *Sequence-like* pour le jeu de Go. Dès lors que la dynamique d'apprentissage est modifiée autour de l'une de ces politiques, les différentes améliorations apportées sont susceptibles de se répercuter ailleurs comme nous le verrons dans le chapitre suivant.

Les deux dynamiques d'apprentissage suivantes ne sont pas utilisées en pratique mais elles nous permettent ici d'illustrer les dynamiques sur lesquelles nous allons ensuite nous baser. Enfin les améliorations pour la dernière dynamique sont actuellement pour d'autres jeux que le Go. En particulier les améliorations N-Gram et LGRF se prêtent bien à des domaines où il n'est pas possible de bénéficier de connaissances spécifiques au jeu comme par exemple le General Game Playing [10]. Ces améliorations ont tout particulièrement retenu notre attention car elles développent des représentations alternatives à l'arbre de recherche pour les politiques de *roll-out*.

4.3 Positionnement de nos contributions

Les améliorations présentées dans la section précédente portent sur chacune des phases mais aussi les différentes interactions entre leurs composantes : la dynamique d'apprentissage. Les contributions présentées dans ce document sont de nouvelles améliorations pour MCTS. Ces améliorations viennent pallier les limitations spécifiques de la méthode. Dans la section 4.3.1 nous présentons les principaux verrous traités par nos contributions. Nous donnons ensuite un aperçu des deux améliorations proposées dans la section 4.3.2. Chacune de ces contributions modifie les interactions entre les différents composants de la dynamique d'apprentissage de MCTS. Nous présenterons à chaque fois la nouvelle dynamique d'apprentissage induite par ces améliorations.

4.3.1 Verrous traités

L'étude du comportement de MCTS a permis d'identifier de nouveaux verrous à cette méthode que ceux initialement traités par les améliorations présentées en section 4.2. Certains problèmes soulèvent des limitations intrinsèques aussi bien aux représentations mobilisées par MCTS qu'à sa dynamique d'apprentissage. Dans cette section, nous évoquons deux verrous auxquels nous proposons de répondre.

10. Le General Game Playing est un domaine dans lequel le programme a connaissance des règles du jeu juste avant le début de la partie.

4.3.1.1. Redondances et pertes de connaissances

Les arbres de recherche constituent une représentation précise de l'environnement qui introduit, de fait, une forme de redondance de l'information (cf. Définition 3.9). Lors de le phase de *descent*, cette représentation permet de décider sans ambiguïté de la prochaine action mais l'agent sera amené à apprendre à nouveau des connaissances semblables pour des situations légèrement différentes. Il s'agit par exemple de petites séquences capables de résoudre un conflit local ou bien éventuellement de danger imminent. Pour une représentation aussi précise, chacune de ces connaissances devra être apprise à nouveau dans chacune des branches [Dra09]. Par ailleurs d'un tour sur l'autre, seulement une partie de l'arbre de recherche sera conservée : celle qui correspond à l'évolution effective de la partie. L'ensemble des connaissances apprises dans les autres branches de l'arbre sera alors oublié. L'agent est alors condamné à apprendre à nouveau ces connaissances, le cas échéant [Dra09].

Afin de limiter cet éternel ré-apprentissage, il est possible d'influencer la valeur initiale des noeuds à partir de représentations complémentaires, à l'instar du *prior knowledge*. Les représentations alors employées sont floues et s'appliquent à un plus grand nombre d'états différents. Cependant ces représentations sont définies *a priori* et ne tiennent pas compte du contexte actuel de la partie.

Des améliorations comme RAVE partagent la même information apprise sur différentes branches de l'arbre. Toutefois la mise à jour de RAVE porte sur la même représentation : l'arbre de recherche. Par définition, une représentation aussi précise ne permet pas une réutilisation efficace de la connaissance. La même information est dupliquée sur les branches de l'arbre mais n'est pas capitalisée dans une structure commune. D'un tour sur l'autre, les évaluations RAVE des branches écartées sont oubliées au même titre que les autres évaluations.

4.3.1.2. Recherche exclusivement globale

Comme nous l'avons évoqué dans la section 4.1.2.4, les évaluations présentes dans l'arbre de recherche sont calculées pour l'essentiel à partir des récompenses obtenues en fin d'épisode. Ces récompenses sont établies suivant l'issue de la partie : c'est à dire conformément à l'objectif global de l'agent. La politique de *descent* repose sur ces évaluations et s'efforce de poursuivre ce même objectif final. Si une telle politique suffit dans la majorité des cas, elle échoue lorsque la recherche doit être approfondie localement.

L'étude du comportement des programmes de Go a révélé que MCTS gérait difficilement des situations faisant intervenir plusieurs sous-problèmes difficiles à résoudre

[HM13 ; Mül10]. Il s'agit par exemple d'identifier si un groupe de pierres[HM13] peut être capturé ou bien de déceler des pièges tendus par son adversaire [RS11], à cause de l'effet d'horizon (cf. Définition 3.8 p.36). La résolution de chacun de ces problèmes nécessite d'approfondir l'arbre de recherche en particulier.

Dans une certaine mesure, les améliorations présentées proposent d'approfondir la recherche suivant certains critères. Des améliorations comme *prior knowledge* ou les *Group Node* privilégient certains noeuds de l'arbre de recherche au travers de leur valeur initiale pour *prior knowledge*, ou bien en constituant des groupes d'actions spécifiques pour les *Group Node*. Cependant ces critères sont à chaque fois définis *a priori*, et ne sont pas susceptibles d'évoluer suivant la position atteinte au cours de la partie. La dynamique d'apprentissage de MCTS ne le permet pas en l'état actuel.

4.3.2 Dynamique d'apprentissage introduite par nos contributions

Les contributions avancées dans ce travail proposent de traiter chacune des limitations présentées ci-dessus. Dans le même temps, chacune de ces contributions modifient la dynamique d'apprentissage de MCTS à l'image des améliorations présentées dans la section 4.2.2. Dans cette section nous donnons un aperçu des travaux que nous allons traiter dans les prochains chapitres suivant la dynamique d'apprentissage introduite.

4.3.2.1. Première contribution : Orienter la politique de *roll-out* par des représentations complexes extraites de l'arbre

La première contribution présentée dans ce manuscrit propose de guider la politique de *roll-out* à partir de connaissances présentes dans l'arbre de recherche. Comme nous l'avons évoqué précédemment, la connaissance présente dans l'arbre de recherche est en grande partie perdue ou répétée dans différentes branches. Afin de valoriser cette connaissance implicite, nous proposons tout d'abord d'extraire de l'arbre de recherche des représentations généralisables à plusieurs situations. Cette représentation est ensuite réutilisée pour guider la politique de *roll-out*. En d'autres termes, cette contribution propose d'étendre la dynamique introduite par des améliorations comme *roll-out history heuristic* pour des représentations comme celles utilisées par NST ou LGRF. La figure 4.7 résume cette contribution à l'aune de la dynamique d'apprentissage introduite.

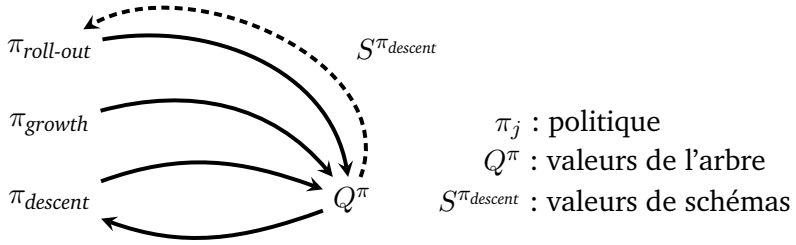


Figure 4.7.– Dynamique d'apprentissage de la première contribution

4.3.2.2. Deuxième contribution : Introduire une méta-dynamique d'apprentissage sur la politique de *descent*

La deuxième contribution présentée dans ce manuscrit propose d'adapter la dynamique d'apprentissage de la phase *descent* suivant l'évolution de la partie. Comme nous l'avons évoqué précédemment, la politique de *descent* poursuit un seul et même objectif global. L'agent est incapable de résoudre des situations requérant une ou plusieurs recherches locales. Afin de pouvoir mettre en oeuvre une recherche locale, nous proposons de changer à la volée de dynamique d'apprentissage pour la phase de *descent*. Il s'agit alors de doter MCTS de la capacité de comparer différentes dynamiques de la phase de *descent*.L'agent serait alors en mesure de pouvoir passer, en temps réel, d'une dynamique à l'autre au travers d'une méta-dynamique d'apprentissage. La figure 4.8 résume la nouvelle dynamique d'apprentissage établie par cette contribution.

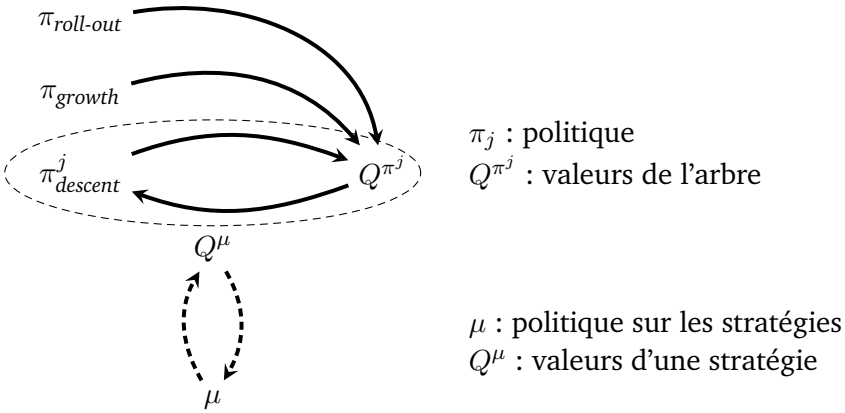


Figure 4.8.– Dynamique d'apprentissage de la deuxième contribution

4.4 Synthèse

Dans ce chapitre, nous avons présenté la méthode MCTS ainsi que les principales améliorations apportées dans l'état de l'art. Nous avons en particulier veillé à détailler chacune des phases de la méthode ainsi que les interactions de ces phases au travers de la dynamique d'apprentissage.

Pour chacune des améliorations nous nous sommes intéressés aux représentations mobilisées. Par ailleurs, nous avons vu que les principales améliorations utilisées actuellement dans les programmes professionnels proposent essentiellement de modifier localement chacune des phases sans remettre en question la dynamique d'apprentissage.

Nous avons ensuite présenté certaines limitations intrinsèques aussi bien au niveau des représentations que de la dynamique d'apprentissage de MCTS. Les contributions présentées dans les chapitres suivants se proposent d'y répondre en introduisant à chaque fois une nouvelle dynamique d'apprentissage pour MCTS.

5

Valorisation des représentations acquises dans la dynamique d'apprentissage

La méthode MCTS apprend en temps réel des évaluations à partir des résultats de simulations aléatoires. Ces évaluations sont accumulées au fur et à mesure des épisodes dans une représentation sous la forme d'un arbre de recherche. En particulier, chaque épisode se déroule en deux temps : une première politique décide des premières actions à partir des valeurs présentes dans l'arbre puis une politique de simulation génère aléatoirement les actions restantes. En l'état, la connaissance emmagasinée dans l'arbre de recherche sert exclusivement la politique dédiée au parcours de cette structure de donnée. Au cours de ce chapitre, nous envisageons de valoriser une partie de cette connaissance présente dans l'arbre pour la politique de simulation. Suivant cette perspective, nous devrons tout d'abord capitaliser cette connaissance dans une représentation plus propice à être réutilisée dans d'autres contextes. À l'image de l'arbre, cette représentation sera progressivement renforcée au cours des simulations. Ainsi, en intégrant cette représentation à la politique de simulation, nous modifions la dynamique d'apprentissage en faisant progressivement évoluer la politique de simulation.

Ce renforcement de la dynamique d'apprentissage constitue la première contribution de cette thèse. Dans la section 5.1, nous revenons plus en détail sur les motivations de ce travail. Nous présentons dans la section 5.2 la principale contribution de ce chapitre : le modèle du *Background History Reply Forest* (BHRF). Dans les sections 5.3 et 5.4 nous exposons deux améliorations possibles de ce modèle. La première porte sur la structure de données utilisée pour accumuler cette connaissance et la seconde propose de simplifier l'évaluation des connaissances accumulées afin d'optimiser la politique de simulation. Dans la section 5.5 nous exposerons les résultats expérimentaux obtenus avec ce modèle pour le jeu de Go. Les modèles et résultats seront ensuite plus longuement commentés dans une dernière partie (cf. section 5.6). Nous comparerons ce modèle avec d'autres approches existantes et formulerons certaines perspectives de recherche pour poursuivre ce travail. Le travail réalisé dans ce chapitre a notamment fait l'objet des publications suivantes [Fab+12 ; Fab+14] ainsi que d'une soumission à un journal [Fab+15].

5.1 Motivations

Les rétroactions fournies par les simulations de MCTS s'inscrivent dans une dynamique d'apprentissage venant progressivement ajuster la politique de l'agent au cours de l'apprentissage. Toutefois la politique utilisée pour générer ces simulations reste par défaut identique tout au long de l'apprentissage. Dans la section 5.1.1, nous revenons sur cet aspect en rappelant succinctement les améliorations existantes dans la littérature. Les connaissances accumulées progressivement dans l'arbre de recherche servent à ajuster la politique associée mais ne sont pas en l'état réutilisable dans d'autres contextes, comme celui des simulations aléatoires. Dans la section 5.1.2, nous rappelons cette limitation intrinsèque de l'arbre de recherche et évoquons comment capitaliser cette connaissance dans une représentation plus propice à sa ré-exploitation.

Une perspective pour adapter la politique de simulation consiste à ré-utiliser des connaissances présentes dans l'arbre de recherche. Les améliorations allant dans ce sens jusqu'alors se limitaient à des représentations élémentaires. Dans la section 5.1.3, nous revenons sur cette perspective en proposant des représentations plus complexes que la simple action. Nous esquissons notamment les contours du modèle du *Background History Reply Forest* (BHRF).

5.1.1 Dynamique supplémentaire à MCTS

Une des principales forces de MCTS réside dans sa capacité à apprendre à la volée les évaluations de nombreux états. Suivant cette perspective, cette dynamique intrinsèque gagne à être étendue afin d'améliorer les performances de la méthode. Nous montrons dans un premier temps que certaines composantes de la méthode restent néanmoins statiques tout au long de l'apprentissage comme la politique de *roll-out*. À cet effet, de nombreuses contributions ont été proposées dans la littérature pour rendre cette politique dynamique. Nous reviendrons ensuite brièvement sur ces contributions dans un second temps.

5.1.1.1. Dynamique d'apprentissage partiellement statique

La dynamique d'apprentissage restitue les interactions entre les processus de résolution et les représentations apprises. À propos de MCTS, nous avions mis en avant dans le chapitre 4 qu'elle comportait 3 politiques : les politiques $\pi_{descent}$, $\pi_{roll\text{-}out}$ et π_{growth}. Ces trois politiques contribuent à renforcer les valeurs présentes dans l'arbre de recherche. Parmi ces trois politiques, la seule à bénéficier d'un ajustement en retour est la politique $\pi_{descent}$ comme cela est présenté dans la figure 5.1.

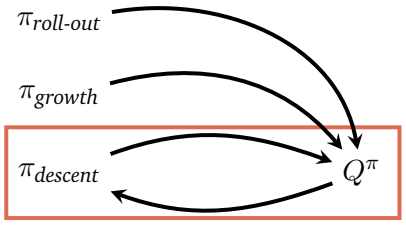


Figure 5.1.– Dynamique d'apprentissage de MCTS

À défaut de bénéficier d'un ajustement, les politiques $\pi_{roll\text{-}out}$ et π_{growth} demeurent identiques tout au long de la partie, quelle que soit la situation courante du jeu. En particulier, la politique $\pi_{roll\text{-}out}$ participe en grande partie à la qualité du renforcement apportée aux valeurs de l'arbre en générant les actions de la simulation aléatoire. Comme nous l'avons évoqué dans la section 4.2.1.3, de considérables gains de performances ont été obtenus par l'amélioration de cette politique au travers de représentations complémentaires, dites *sequence-like* [WG07]. Toutefois les représentations employées dans les programmes actuels ne bénéficient d'aucune rétroaction au cours de l'apprentissage : les politiques dépendantes restent statiques tout au long de la partie.

5.1.1.2. Politique de *roll-out* adaptative

Les politiques de *roll-out* adaptatives, au contraire, sont en mesure d'ajuster progressivement leur comportement au travers de représentations complémentaires bénéficiant d'une rétroaction. Dans le chapitre précédent, nous avons vu que la majorité de ces politiques proposent de construire une représentation complémentaire à l'arbre de recherche et indépendante de celui-ci (cf. section 4.2.2.3). Les renforcements perçus par ces nouvelles représentations correspondent aux actions sélectionnées au cours de la phase de *roll-out* et non de la phase de *descent* comme pour l'arbre de recherche.

L'utilisation de connaissances présentes dans l'arbre de recherche a été déjà évoquée dans la littérature [Bas+14] pour influencer la politique de *roll-out*. Cependant les seules améliorations se limitent à des représentations élémentaires comme nous l'avons présenté dans la section 4.2.2.2. Des représentations aussi simples sont susceptibles d'introduire un important biais au cours de la simulation.

5.1.2 Capitaliser la connaissance de l'arbre de recherche

Un arbre de recherche constitue une représentation très précise qu'il est difficile de réutiliser dans d'autres contextes. D'un tour sur l'autre, une grande partie de

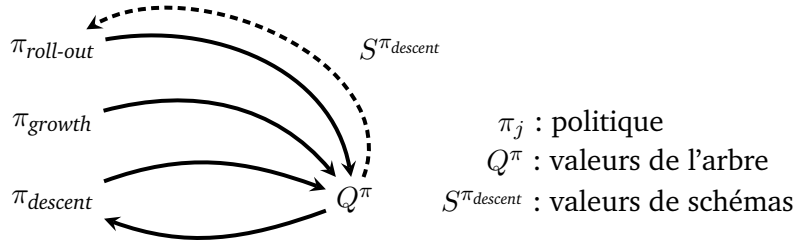


Figure 5.2.– Dyanmique d'apprentissage avec BHRF

cette connaissance apprise est oubliée par l'agent pour de nouvelles représentations spécifiques à la nouvelle situation. L'agent est ainsi en perpétuel ré-apprentissage des mêmes connaissances dès lors que les états associés diffèrent d'une seule action. Cette limitation, que nous avons plus amplement détaillée dans la section 4.3.1.1, est intrinsèque de la représentation par arbre de recherche, retenue pour accumuler cette connaissances au cours de l'apprentissage.

La réutilisation des connaissances présentes dans l'arbre de recherche requiert une représentation généralisable à un plus grand nombre de situations. Comme l'arbre de recherche est une représentation essentiellement temporelle, une représentation relativement proche pour capitaliser ces informations consiste à employer des schémas temporels (cf. Définition 3.10 à la page p. 3.10). Chaque branche ou sous-branche de l'arbre de recherche est une séquence d'actions et par conséquent assimilable à un schéma temporel. L'abstraction alors réalisée est essentiellement par restriction, c'est à dire qu'elle traite un plus grand nombre de situations en ne s'intéressant qu'à une partie de la description d'un état.

5.1.3 Valoriser les connaissances de l'arbre dans la dynamique d'apprentissage

Dans MCTS, les connaissances présentes dans l'arbre de recherche servent exclusivement à la politique $\pi_{descent}$ au cours de l'apprentissage. Avec cette contribution, nous proposons d'extraire de l'arbre de recherche des connaissances pour améliorer la politique $\pi_{roll\text{-}out}$. Dans le cadre de la Background History Reply Forest (BHRF), ces connaissances seront extraites sous la forme de schémas temporels, à l'instar d'autres politiques adaptatives proposées dans la littérature. Cependant la provenance des représentations mobilisées crée une nouvelle boucle dans la dynamique d'apprentissage comme cela est présenté dans la figure 5.2. Cette interdépendance entre la politique $\pi_{roll\text{-}out}$ et les valeurs présentes dans l'arbre Q^{π} reste toutefois moins directe que celle entre la politique $\pi_{descent}$ et ces mêmes valeurs.

Afin d'extraire les connaissances accumulées dans l'arbre de recherche dans une autre représentation, la démarche la plus efficace consiste à renforcer cette représentation avec les mêmes rétroactions que celles perçues par l'arbre de recherche. À l'image de l'arbre de recherche, cette représentation devra être progressivement étendue. C'est pourquoi le choix de la structure de données retenues pour accumuler ces connaissances sera décisif, au même titre que les modifications à apporter aux phases *update* et *growth* pour la renforcer.

5.2 Background History Reply Forest

La Background Reply History Forest (BHRF) est un processus d'apprentissage par renforcement complémentaire de MCTS pour améliorer sa politique de simulation. En particulier, cette contribution vise à valoriser les connaissances de l'arbre construit par MCTS sous la forme de schémas temporels. Ces schémas temporels sont appris dans une représentation complémentaire à l'arbre de recherche à partir des mêmes rétroactions. Cette représentation complémentaire est ensuite utilisée lors de la politique de simulation afin de l'adapter au contexte actuel de la partie.

Dans la section 5.2.1 nous présentons la représentation complémentaire que nous avons conçue pour les besoins de BHRF. Dans la section 5.2.2 nous présentons les modifications à apporter aux phases *update* et *growth* pour renforcer cette structure de données conjointement à l'arbre de recherche. Enfin nous précisons comment intégrer cette représentation complémentaire à la politique de *roll-out* dans la section 5.2.3.

5.2.1 Représentation par forêt rétrograde de schémas

Les représentations présentes dans l'arbre de recherche portent systématiquement sur des descriptions exhaustives des états et ne sont pas réutilisables dans d'autres contextes. Par ailleurs, les connaissances accumulées par l'arbre de recherche augmente à mesure que celui-ci est étendu au cours des épisodes. Ainsi, la représentation complémentaire nécessaire à BHRF pour réexploiter les connaissances de l'arbre doit répondre à un double impératif :

— une représentation plus propice à sa réutilisation ;

— une structure de données progressivement extensible.

Dans cette section nous présentons une nouvelle structure de données appelée forêt rétrograde de schémas, qui constitue la représentation la plus immédiate pour répondre à ces deux impératifs. Nous motiverons tout d'abord l'élaboration de cette structure de données jumelle de l'arbre de recherche en revenant plus en détails sur

le double impératif évoqué. Ensuite nous décrirons plus formellement les différentes composantes de cette structure de données.

5.2.1.1. Motivations

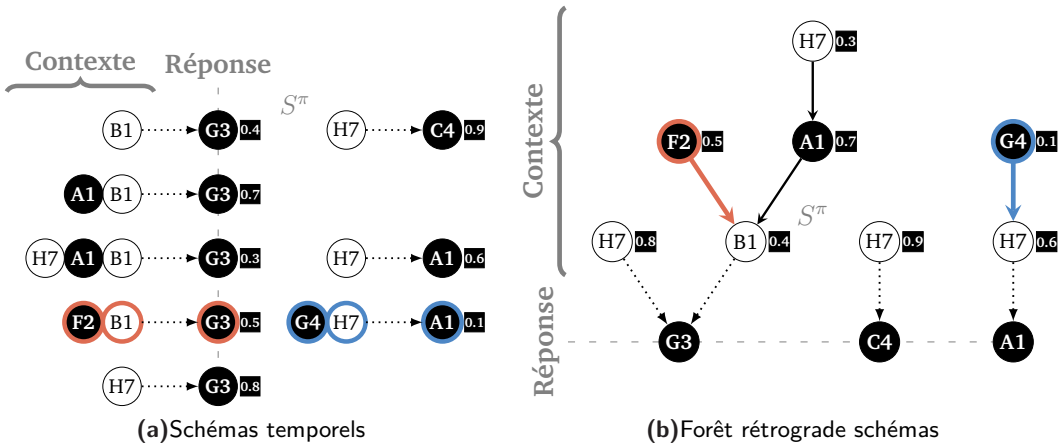
Un arbre de recherche organise des représentations d'états de l'environnement suivant un ordre chronologique (cf. Définition 3.7 p. 35). Il ne couvre en général qu'une partie de l'espace de recherche. Toutefois la structure même d'un arbre de recherche laisse toujours l'opportunité de l'étendre progressivement. Par exemple MCTS intègre, à la fin de chaque épisode, des états immédiatement consécutifs à un état déjà couvert par l'arbre de recherche, c'est à dire suivant le sens chronologique. Cette construction incrémentale permet d'approfondir notamment avec parcimonie l'espace de recherche : une zone sera plus détaillée si plusieurs épisodes successifs poussent à son exploration. L'agent gagnera en anticipation à mesure que l'arbre de recherche sera étendu.

La forêt de schéma rétrograde propose de capitaliser les connaissances présentes dans l'arbre de recherche par des schémas temporels, plus facilement réutilisables dans d'autres contextes. Un schéma temporel est une séquence d'actions consécutives dont les premières correspondent au contexte et la dernière à la réponse (cf. Définition 3.10 p. 37). Cette représentation temporelle est assez immédiate pour extraire des connaissances de l'arbre. En effet il est tout à fait possible de considérer chaque noeud de l'arbre de recherche comme un schéma temporel pour la situation courante : le contexte correspond à la séquence d'actions reliant la position courante au noeud parent et la réponse correspond à l'action séparant le noeud parent du noeud fils. Suivant cette perspective, chaque nouvel état ajouté à l'arbre de recherche revient à considérer un nouveau schéma dont le contexte correspond à un schéma précédemment couvert par l'arbre, mais la réponse correspond à l'action nouvellement ajoutée en suffixe.

À l'image de l'arbre de recherche, la forêt de schéma rétrograde laisse l'opportunité de s'étendre progressivement. Cependant, l'extension de cette structure ne servira pas à anticiper les évolutions futures mais à préciser davantage les schémas actuellement couverts par la structure en augmentant le contexte associé à chaque réponse. Afin de garantir une certaine parcimonie dans les représentations couvertes, la structure sera construite incrémentalement, c'est à dire que l'ensemble des représentations ajoutées seront progressivement étendues à partir de celles précédemment couvertes. Ainsi, un schéma déjà couvert sera étendu d'une seule action au niveau du préfixe de son contexte : en toute première action. Cette construction semblable à celle de l'arbre de recherche se distingue toutefois par l'ordre suivant lequel les actions sont considérées. Dans l'arbre de recherche, les représentations nouvellement ajoutées considèrent des

actions survenues plus tard dans le temps : sens direct ou chronologique. À l'inverse dans la représentation proposée, les représentations ajoutées considèrent des actions survenues plus tôt dans le temps : sens antichronologique ou rétrograde.

5.2.1.2. Description de la structure de données



(a) Schémas temporels **(b)**Forêt rétrograde schémas

Figure 5.3.– Organisation des schémas temporels dans une forêt rétrograde de schémas.

Une forêt de schémas temporels agrège un ensemble de schémas temporels comme cela est présenté dans la figure 5.3. Dans la théorie des graphes, une forêt correspond à un ensemble d'arbres à l'image de l'arbre de recherche. En l'occurrence, chaque arbre de la forêt est associé à une réponse possible. La racine de l'arbre indique la réponse à apporter et chaque noeud de l'arbre indique une action ayant précédé cette réponse. Le contexte du schéma temporel considéré correspond au chemin reliant le noeud à la racine. C'est pourquoi les arcs ainsi que l'arbre sont dessinés à l'envers du sens usuel. La valeur associée à un noeud indique alors la valeur associée au schéma temporel comprenant le chemin jusqu'à la racine (contexte) et la racine elle-même (réponse). Du fait de sa structure, la forêt rétrograde de schéma vérifie en permanence la propriété suivante :

Propriété 5.1

Pour tout schéma temporel porté par un noeud d'une forêt rétrograde de schéma, tous les suffixes de la séquence d'actions correspondant à ce schéma existent dans la forêt sous la forme de schémas plus petits. Il s'agit de la branche reliant ce noeud à la racine.

En principe cette structure de données n'est pas limitée en profondeur à l'image de l'arbre de recherche. Cependant, pour des questions de mémoire nous proposons de fixer la profondeur maximale de chacun des arbres par le paramètre *depth*. En

l'occurrence, la profondeur maximale signifie aussi la taille maximale du contexte considéré pour chaque schéma temporel couvert.

5.2.2 Mise à jour de la représentation

Les connaissances apprises dans l'arbre de recherche dépendent des seules récompenses progressivement perçues au cours des épisodes comme nous l'avons vu dans le chapitre 4. Ainsi la meilleure approche pour extraire des connaissances présentes dans l'arbre de recherche consiste à faire bénéficier notre représentation complémentaire des mêmes rétroactions. Cependant, cette rétroaction ne sera pas interprétée de la même manière suivant la structure de données considérée. En effet les représentations couvertes par l'arbre de recherche n'induisent aucune ambiguïté dans leur interprétation, à la différence des schémas temporels considérés par BHRF.

Dans cette section, nous préciserons comment interpréter les rétroactions fournies à l'arbre de recherche afin de capitaliser dans la forêt rétrograde de schémas des connaissances notamment présentes dans celui-ci. Dans un premier temps, nous évoquerons les modifications à apporter aux phases d'*update* et *growth* pour faire évoluer conjointement la forêt rétrograde de schémas et l'arbre de recherche. Ensuite nous discuterons de la complexité de la mise à jour BHRF ainsi que certaines propriétés induites sur les schémas couverts par la forêt rétrograde.

5.2.2.1. Modifications apportées aux phases *update* et *growth*

Dans la méthode MCTS, l'arbre de recherche est modifié au cours des phases *update* et *growth*. Lors de la phase *update*, les états couverts par l'arbre de recherche sont renforcés par la récompense obtenue en fin d'épisode. Les noeuds mis à jour correspondent aux états qui ont été traversés par la séquence d'actions jouée au cours de la phase *descent*. Lors de la phase *growth*, un nouvel état est ajouté à l'arbre de recherche. Cet état est atteint à exactement une action de plus qu'un état précédemment couvert par l'arbre de recherche.

Dans BHRF, la forêt rétrograde de schémas bénéficie de la même rétroaction que celle de l'arbre de recherche : la séquence d'action sélectionnée au cours de la phase de *descent* et la récompense obtenue à la fin de l'épisode. De plus, cette rétroaction sera interprétée d'une manière similaire à celle de MCTS à la différence près que la représentation renforcée considère ici des schémas temporels au lieu d'états du jeu. Comme l'arbre de recherche, la forêt de schémas rétrograde sera renforcée au

cours des phases *update* et *growth* [1]. Les algorithmes détaillés sont consultables dans l'Annexe A.4.

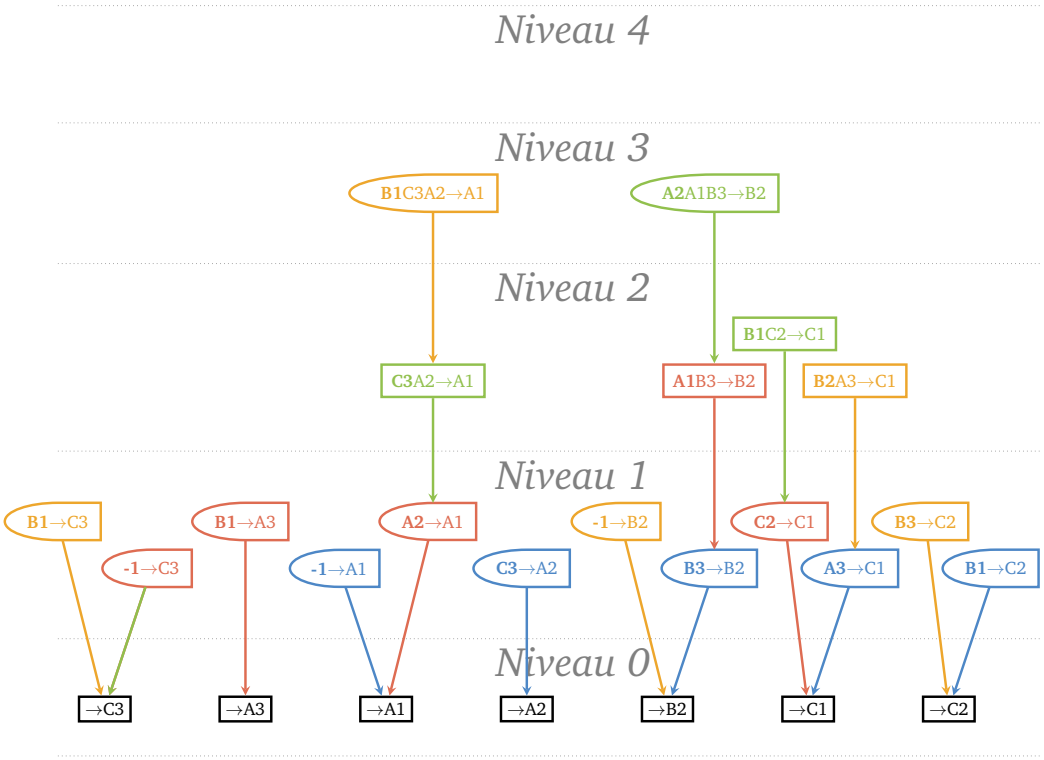
Lors de la phase *update* les schémas temporels couverts par la forêt rétrograde de schémas sont renforcés à partir du résultat de la simulation. Les noeuds sont mis à jour s'ils correspondent à un schéma temporel présent dans la séquence des actions jouées au cours de la phase *descent*. Pour une même réponse jouée dans la séquence de *descent*, tous les schémas dont le contexte coïncide avec les actions précédemment jouées sont renforcés ; c'est à dire les noeuds situés sur la même branche d'un arbre associé à une réponse. Aucun schéma temporel n'est ajouté pour une réponse jouée si le schéma remontant jusqu'à la racine de l'arbre de recherche est déjà couvert par l'arbre associé.

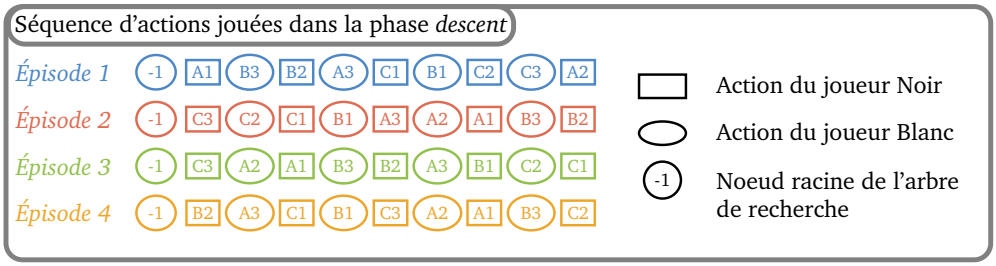
Lors de la phase *growth*, un nouveau schéma temporel est ajouté en principe pour chaque action jouée au cours de la phase *descent* : un schéma temporel par arbre dont la réponse a été jouée. Ce schéma nouvellement ajouté correspond à une séquence d'actions comportant une action par rapport à un autre schéma couvert, au niveau du préfixe ; en d'autres termes, un nouveau noeud feuille est ajouté à l'arbre associé à cette même réponse. Cette construction incrémentale de la forêt limite son extension à des schémas de grande taille. En effet l'ajout d'un nouveau temporel requiert, par construction, que tous les schémas plus petits et avec les mêmes dernières actions aient été préalablement ajoutés à la forêt, lors d'épisodes précédent.

La figure 5.4 illustre la construction de la forêt rétrograde de schémas sur plusieurs mises à jour successives. La forêt initialement vide est progressivement étendue à la fin de chaque mise à jour suivant l'algorithme de construction proposé. Cet exemple nous permet notamment d'illustrer l'accumulation incrémentale de nouveaux schémas. Ainsi, bien que la séquence A2-A1-B3-B2 apparaisse dès le deuxième épisode, le schéma temporel correspondant ne sera pas ajouté à cet épisode-ci car la forêt ne couvrait pas encore le schéma temporel A1-B3→B2 (qui est ajouté à cet épisode). Il faudra attendre que cette même séquence d'actions soit répétée dans un épisode suivant pour ajouter le schéma A2-A1-B3→B2.

L'extension par BHRF de le forêt de schémas est similaire à celle de l'arbre de recherche dans MCTS. Cependant les schémas ainsi accumulés par BHRF ne recouvrent pas nécessairement la totalité des schémas présents dans l'arbre de recherche car les méthodes considèrent des extensions des représentations suivant deux sens chronologiques opposés (anti-chronologique pour BHRF et chronologique pour MCTS). Dans l'Annexe A.3 nous proposons une illustration de ce phénomène pour un exemple simplifié.

1. En pratique, les phase de *update* et *growth* sont réalisées pour BHRF simultanément en fin d'épisode.


Niveau 4
Niveau 3
B1C3A2→A1
A2A1B3→B2
Niveau 2
B1C2→C1
C3A2→A1
A1B3→B2
B2A3→C1
Niveau 1
B1→C3
B1→A3
A2→A1
-1→B2
C2→C1
B3→C2
-1→C3
-1→A1
C3→A2
B3→B2
A3→C1
B1→C2
Niveau 0
→C3
→A3
→A1
→A2
→B2
→C1
→C2


Séquence d'actions jouées dans la phase descent
Épisode 1
-1 A1 B3 B2 A3 C1 B1 C2 C3 A2
Épisode 2
-1 C3 C2 C1 B1 A3 A2 A1 B3 B2
Épisode 3
-1 C3 A2 A1 B3 B2 A3 B1 C2 C1
Épisode 4
-1 B2 A3 C1 B1 C3 A2 A1 B3 C2
Action du joueur Noir
Action du joueur Blanc
-1
Noeud racine de l'arbre de recherche

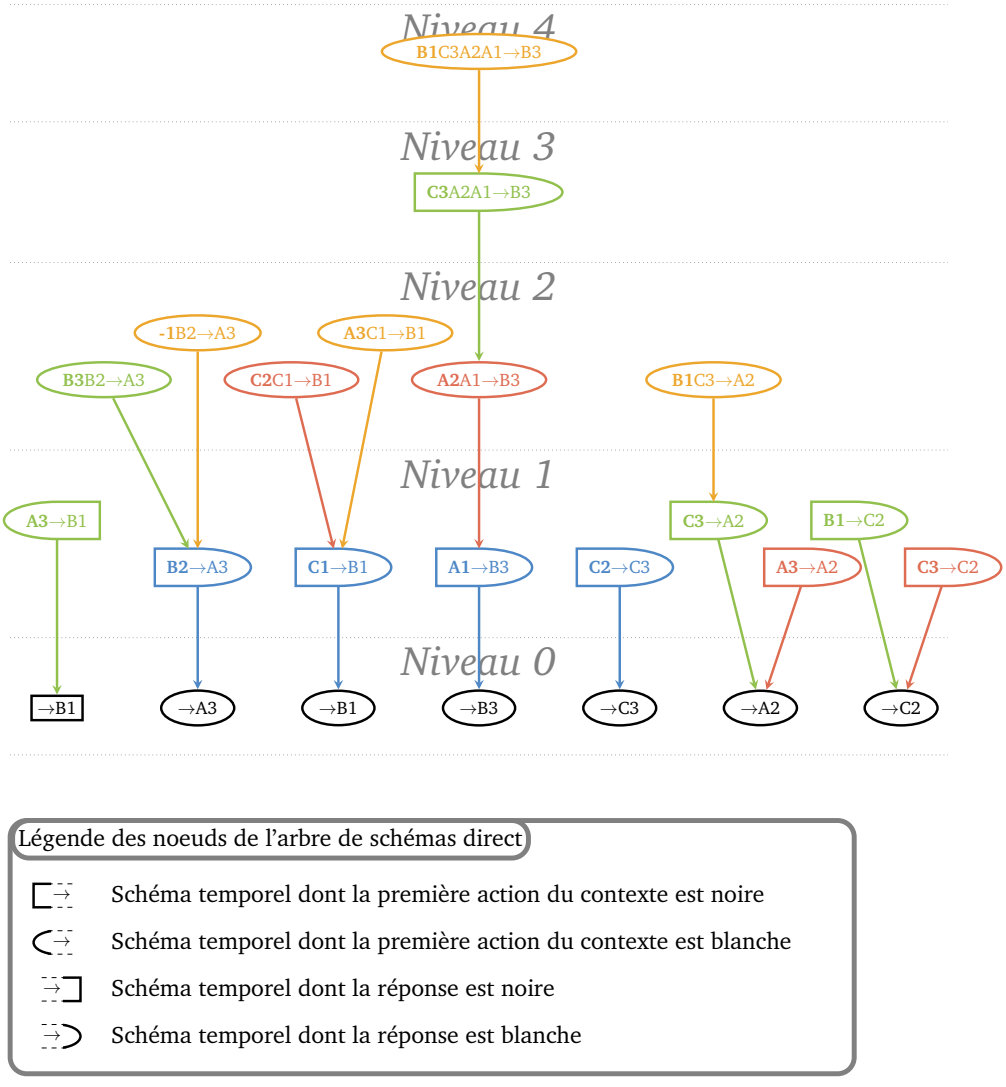


Figure 5.4.– Construction incrémentale d'une forêt rétrograde de schémas

5.2.2.2. Complexité de la mise à jour et propriété induite

Si l'arbre de recherche et la forêt rétrograde de schémas sont mis à jour à partir de la même rétroaction, la complexité algorithmique de ce renforcement est bien différente pour les deux structures. Dans MCTS, un seul état en plus est considéré à chaque épisode. Chaque état parcouru lors de la phase de *descent* est mis à jour une fois. La complexité totale de la mise à jour est d'un ordre de grandeur de $O(d)$ (avec d la taille de la séquence de *descent*). Dans BHRF, plusieurs schémas temporels sont susceptibles d'être ajoutés lors de la mise à jour. Chaque action jouée dans la phase de *descent* représente potentiellement la réponse d'un nouveau schéma. De plus chaque schéma inclus dans le nouveau schéma ajouté est aussi mis à jour. Ainsi la complexité de la mise à jour est ici d'un ordre de grandeur de $O(d^2)$. Cette complexité est bien plus grande que celle de MCTS. Néanmoins cette expansion incrémentale de la forêt pour toutes les réponses jouées successivement dans la phase de *descent*, induit la propriété suivante :

Propriété 5.2

Soit une forêt de schémas temporels renforcée par la mise à jour BHRF. Pour tout schéma temporel s porté par le noeud d'une forêt rétrograde de schéma, toute séquence d'actions consécutives présente dans ce schéma est présente dans la forêt sous la forme d'un schéma plus petit.

Voici une démonstration sommaire de cette propriété. Conformément à la définition 3.10 p.37, un schéma temporel est équivalent à une séquence d'actions ordonnée dont la réponse est située en position n et le contexte correspond aux autres positions allant de 1 à $n-1$. S'il existe à la fin d'un épisode un schéma s de taille n dans cette forêt, cela signifie que la réponse s_n de ce schéma a bénéficié de $n-1$ renforcement couvrant progressivement tous les schémas pour des contextes de taille croissante, suivant le sens rétrograde. Dans le même temps, cela signifie qu'ailleurs dans la forêt, les réponses associées aux actions de ce contexte seront aussi étendues à partir des mêmes séquences d'actions. Par exemple l'arbre associé à l'action s_{n-1} aura bénéficié d'au moins $n-2$ renforcements à partir des mêmes séquences d'actions que celles utilisées pour renforcer s_n. Ainsi tous les contextes allant de s_{n-2} à s_1 seront aussi couverts par l'arbre associé à s_{n-1}. En appliquant ce même raisonnement pour toutes les actions présentes dans le contexte de s, cela signifie qu'il existe ailleurs dans la forêt, un schéma correspondant à toute séquence d'action présente dans le contexte de s.

Cette propriété nous servira notamment à élaborer une structure de donnée optimisée pour stocker les schémas couverts par BHRF présenté dans la section 5.3.

5.2.3 Politique de *roll-out* avec BHRF

Les connaissances capitalisées par BHRF servent à améliorer la politique de *roll-out*. Les schémas temporels couverts par la forêt rétrograde de schémas sont intégrés à la politique de *roll-out* par défaut. Cependant, la diversité des schémas couverts par cette structure de données complémentaire soulèvent plusieurs questions à propos de leur réutilisation. En particulier les schémas temporels accumulés possèdent chacun des contextes de tailles différentes et des évaluations plus ou moins fiables suivant le nombre de renforcements perçus. Chacun de ces aspects introduit un biais dans la sélection des actions que nous avons choisi ici de considérer séparément.

Dans la section 5.2.3.1, nous présentons les principales étapes de la politique de *roll-out* avec BHRF. Dans les sections 5.2.3.2 et 5.2.3.3, nous revenons plus en détail sur les choix réalisés pour considérer, respectivement, la taille des contextes et la fiabilité de leur évaluation au travers de la probabilité qui leur est associée. Les améliorations proposées dans la suite de ce chapitre porteront, pour l'essentiel, sur chacun de ces deux aspects de la politique de *roll-out* respectivement dans les sections 5.3 et 5.4.

5.2.3.1. Processus de sélection des réponses

La politique BHRF vient compléter la politique de *roll-out* par défaut de MCTS. Par cette politique, les actions du *roll-out* sont sélectionnées à partir des réponses préconisées par les schémas accumulés dans la forêt rétrograde. La réponse adéquate est sélectionnée suivant un processus que nous pouvons séparer en trois étapes :

1. Sélection déterministe des schémas ;
2. Sélection stochastique de la réponse ;
3. Application stochastique de la réponse.

La politique BHRF décrite par ces trois étapes est appliquée avec une probabilité ϵ fixée *a priori*, afin de réguler son utilisation. Lorsque cette politique n'est pas appliquée ou ne suggère aucune réponse, la politique de *roll-out* par défaut est ensuite appliquée. Ainsi, la proportion effective d'actions générées par BHRF est strictement inférieure à la valeur ϵ puisque la procédure de sélection d'un schéma n'aboutit pas nécessairement.

L'algorithme de la politique de *roll-out* décrite dans cette section est consultable dans l'annexe A.5. Dans les paragraphes suivant nous allons revenir plus en détails sur les trois étapes de ce processus. Le déroulement général du processus de sélection de réponse est résumé par la figure 5.5.

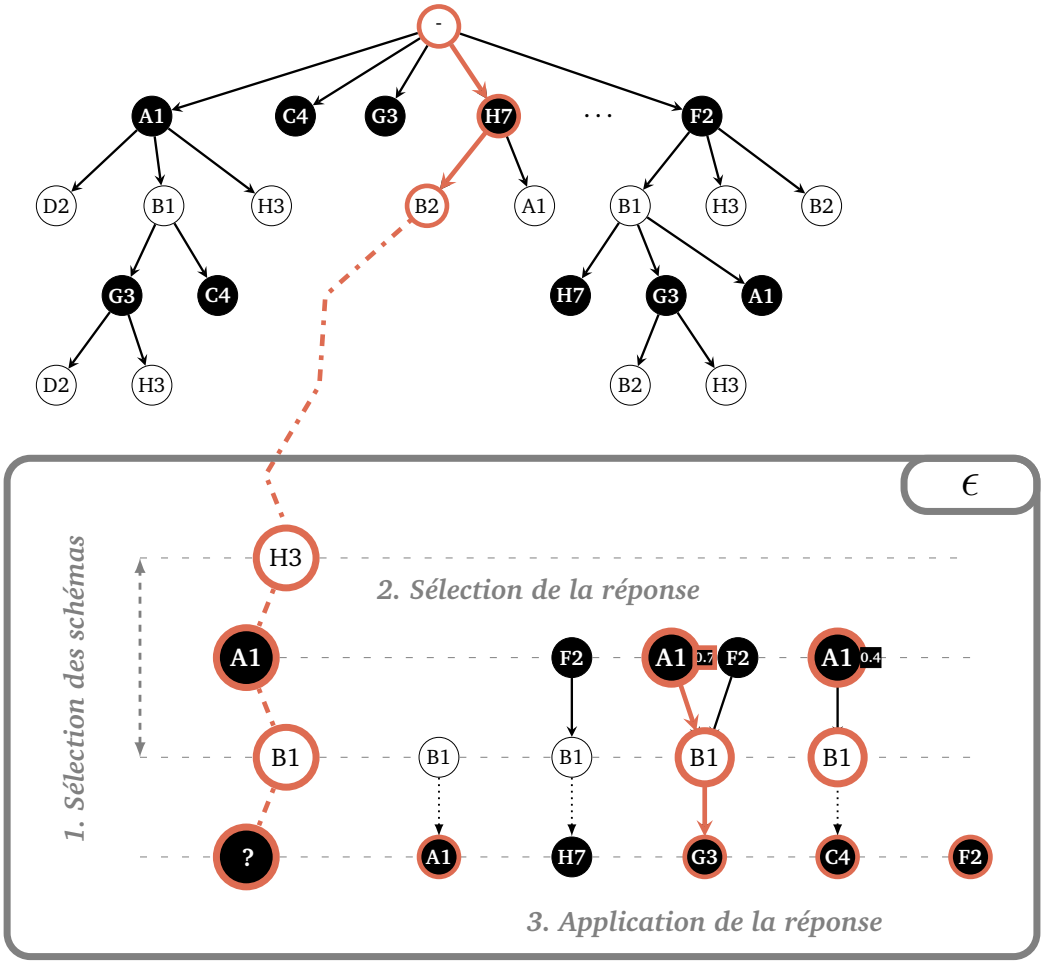


Figure 5.5.– Processus de sélection d'action de BHRF : dans un premier temps les schémas temporels dont le contexte coïncide avec le plus grand nombres d'actions précédemment jouées sont conservés, ensuite la réponse adéquate est choisie à partir les évaluations des schémas retenus. Enfin la réponse sélection est appliquée avec une probabilité dépendant de cette évaluation.

Sélection déterministe des schémas Dans un premier temps, les schémas présents dans la forêt rétrograde sont filtrés suivant leur contexte. En particulier les schémas conservés correspondent à ceux dont le contexte coïncide avec le plus grand nombre d'actions précédemment jouées et la réponse est légale pour la situation de jeu. Nous reviendrons sur les motivations d'un tel filtre dans la section 5.2.3.2.

Sélection stochastique de la réponse Ensuite une réponse est sélectionnée aléatoirement parmi celles associées aux schémas conservés. Pour chaque réponse une probabilité est calculée à partir des évaluations associées aux schémas. Nous reviendrons plus en détail sur le calcul de cette probabilité dans la section 5.2.3.3. Une seule réponse est sélectionnée pour participer à la dernière étape, par une roulette biaisée à partir des probabilités établies.

Application stochastique de la réponse Enfin la réponse sélectionnée est effectivement appliquée suivant la probabilité calculée lors de l'étape précédente. En effet, à cause de la première étape, toutes les réponses possibles ne sont pas nécessairement présentes lors de la sélection stochastique. Cet ultime filtre permet de limiter l'application systématique de la réponse, notamment dans les cas où la roulette biaisée portait sur un nombre très restreint de réponses candidates.

5.2.3.2. Sélection des schémas suivant leur contexte

Un schéma temporel est applicable à une situation de jeu lorsque les dernières actions jouées coïncident exactement avec le contexte du schéma et la réponse proposée est légale pour cette situation. Parmi les schémas couverts dans la forêt rétrograde, plusieurs d'entre eux sont applicables à une même situation de jeu, suivant qu'ils proposent des réponses différentes pour un même contexte ou bien des contextes de tailles différentes pour une même réponse légale. Dans la première étape de la politique BHRF, nous avons choisi de ne considérer que des schémas temporels applicables avec un contexte identique et le plus grand possible.

Tout d'abord, nous n'avons retenu que des schémas avec un contexte identique afin de limiter justement l'influence du contexte lors de la sélection de la réponse. En effet, lorsque les contextes sont différents, l'évaluation du schéma temporel ne porte pas seulement sur la pertinence de la réponse pour le contexte mais aussi sur la pertinence de contexte lui-même pour la situation. En particulier, lorsque les évaluations sont apprises à la volée, la comparaison des schémas avec des contextes différents peut privilégier certaines réponses simplement parce que le contexte est apparu plus souvent dans des situations favorables [Lew11] [2].

Ensuite les contextes des schémas retenus sont les plus grands afin d'avoir la description la plus précise possible de l'état courant. Dans une certaine mesure, la politique BHRF simule un arbre de recherche local pour la phase de *roll-out*. Le noeud atteint ne correspond pas à la description exhaustive d'un état comme dans un arbre de recherche mais au plus grand contexte temporel disponible pour décrire cet état. Les réponses des schémas retenus sont autant d'actions possibles depuis ce noeud qui seront ensuite départagées lors de la deuxième étape.

En pratique, la sélection des schémas avec le plus grand contexte possible représente une étape très coûteuse en temps de calcul pour une forêt rétrograde de schémas : chaque arbre associé à une réponse légale est remonté en suivant les actions précédemment jouées. Dans la section 5.3, nous proposons une structure de données

2. Lew a décrit ce phénomène pour des schémas spatiaux dans son mémoire de doctorat[Lew11] : suite à une action décisive comme la capture d'une grande chaîne au Go, de nombreuses mauvaises réponses étaient ensuite choisies à proximité de la zone capturée, simplement pour leur contexte.

différente de la forêt rétrograde de schémas afin d'optimiser le temps consacré à cette étape de la politique.

5.2.3.3. Probabilité associée aux réponses pour un même contexte

Les évaluations des schémas temporels de BHRF proviennent de l'arbre de recherche. Certaines évaluations de l'arbre de recherche sont incertaines car elles sont calculées pour un trop faible échantillon de simulations. Pour la phase *descent*, l'amélioration UCT concilie efficacement l'évaluation d'une action avec sa fiabilité à chaque étape de la recherche (cf. section 4.2.1.1). Suivant cette perspective, nous avons choisi d'adapter UCT pour une sélection stochastique des schémas temporels. Comme nous l'avons évoqué précédemment, la politique BHRF revient à construire momentanément un arbre local. Ainsi les probabilités des schémas considérés lors de la deuxième étape sont calculées pour un même contexte comme suit :

$$P(r|c) = \frac{S^{\pi_{descent}}(c,r) + b \times \sqrt{\frac{\ln \sum_{i \in \mathscr{C}} n_{i|c}}{n_{r|c}}}}{\sum_{i \in \mathscr{C}} P(i|c)}, \tag{5.1}$$

tel que

r	: réponse légale
c	: contexte
$\mathscr{C}$	: ensemble des réponses légales pour le contexte c
$S^{\pi_{descent}}(c,r)$	: valeur moyenne de la réponse r pour le contexte c
b	: constante UCT
$n_{r\|c}$	: nombre de renforcement du schéma temporel

Ainsi la probabilité de sélectionner puis d'appliquer une réponse dépend de l'évaluation du schéma $S^{\pi_{descent}}(c,r)$ mais aussi du nombre de renforcement de l'ensemble des schémas retenus pour le calcul de la valeur UCT. Cette valeur UCT permet de concilier la valeur moyenne du schéma avec sa fiabilité. Toutefois ce calcul est bien plus coûteux en temps que des politiques *softmax* suivant une distribution de Gibbs par exemple. Dans la section 5.4, nous proposerons de simplifier le calcul de ces probabilités afin d'optimiser le temps de calcul nécessaire à cette étape de la politique BHRF.

5.3 Optimisation de la structure de données

Comme nous l'avons vu dans la section 5.2.1.1, la forêt rétrograde de schémas a été conçue pour accumuler des schémas temporels de taille croissante. En particulier, les nouveaux schémas considérés possèdent une action en plus au niveau du préfixe d'un schéma déjà couvert. Si la structure rétrograde de cette forêt se prête bien à une accumulation incrémentale des nouveaux schémas considérés, elle ne favorise pas leur réutilisation lors de la politique de *roll-out*. En effet, nous avons évoqué dans la section 5.2.3.2 que la première étape de la politique BHRF est très coûteuse en temps de calcul pour une forêt rétrograde de schémas.

La forêt rétrograde de schémas n'est pas nécessaire à la réalisation de BHRF mais elle nous semble plus appropriée pour expliquer le mécanisme d'accumulation de nouveaux schémas. Dans cette partie, nous présentons une autre structure de données plus appropriée pour la politique d'apprentissage : un arbre direct de schémas. Les schémas temporels présents dans cette structure sont exactement les mêmes que ceux couverts dans la forêt rétrograde de schéma. Les schémas sont simplement organisés différemment en mémoire afin de faciliter leur réutilisation. Nous nous intéressons en particulier aux questions de complexité algorithmique à propos de la mise à jour de la structure de données ainsi que son exploitation dans la politique BHRF.

Dans la section 5.3.1, nous revenons sur les motivations, en terme de complexité, pour élaborer une nouvelle structure de données. Nous présentons plus en détail la nouvelle structure de données dans la section 5.3.2. Ensuite nous détaillons la mise à jour de cette représentation ainsi que son intégration à la politique de *roll-out* dans les sections 5.3.3 et 5.3.4 respectivement.

L'organisation des schémas temporels accumulés par BHRF n'introduit aucune différence de fond avec le modèle général présenté dans la section 5.2. Il s'agit exclusivement d'une optimisation algorithmique. Aussi, les contributions proposées ne sont pas indispensables pour comprendre les résultats obtenus ensuite grâce à BHRF. Le lecteur peut librement se dispenser de la lecture de cette partie, dans un premier temps.

5.3.1 Motivations pour une structure optimisée

Les schémas temporels accumulés par BHRF sont ensuite réutilisés lors de la phase de *roll-out* au travers d'un processus spécifique que nous avons plus longuement décrit dans la section 5.2.3. La première étape de ce processus consiste à sélectionner parmi les schémas temporels de BHRF, ceux dont la réponse est légale et dont le

contexte coïncide avec le plus d'actions précédentes. Comme nous l'avions évoqué dans la section 5.2.3.2, il s'agit de simuler localement un arbre de recherche pour le plus grand contexte possible. Cette opération pour la forêt rétrograde de schémas peut s'avérer très coûteuse en temps de calcul pour une situation donnée. En outre, ce processus de sélection est susceptible d'être appliqué à plusieurs états successifs de la phase de *roll-out* multipliant d'autant le ralentissement occasionné.

Dans un premier temps, nous reviendrons sur la complexité algorithmique de cette étape du processus pour la forêt rétrograde. Nous esquisserons ensuite les contours d'une structure de données capable de réaliser cette opération en un temps constant pour une situation donnée. Cette structure de données sera ensuite précisée dans la suite de cette partie.

5.3.1.1. Complexité de la sélection de schémas pour la forêt rétrograde

Pour la forêt rétrograde de schémas, la méthode la plus immédiate pour récupérer les schémas avec le plus grand contexte nécessite de repartir systématiquement des racines de chacun des arbres. Pour chaque racine correspondant à une réponse légale, l'arbre est parcouru suivant les dernières actions jouées à rebours dans le temps jusqu'à ce que le contexte ne soit plus couvert par l'arbre. Nous obtenons ainsi les schémas de plus grand contexte pour toutes les réponses légales. Il suffit ensuite de ne conserver que ceux correspondant au plus grand contexte dans l'ensemble.

La complexité d'un tel algorithme est de l'ordre de $O(bp)$ dans le pire des cas, où b correspond aux actions légales en moyennes (ou facteur de branchement moyen) et p la profondeur maximale atteinte par les arbres de BHRF. Lorsque la profondeur maximale n'est pas limitée, la complexité maximale par itération est bornée par un ordre de grandeur quadratique $O(b^2)$ dans le pire des cas. En pratique, cette complexité nous a semblé bien inférieure mais ces valeurs mettent en avant l'inefficacité de cette méthode. À chaque nouvelle application de la politique BHRF, toutes les réponses légales doivent être considérées avant d'identifier celles correspondant aux schémas avec le plus grand contexte. Lorsque la politique BHRF est appliqué à chaque étape de la phase de *roll-out*, la complexité dans le pire des cas associée à cette étape est d'un ordre de grande de $0(b^2r)$, avec r le nombre d'état traversé dans la phase de *roll-out*. D'une application à l'autre, l'organisation rétrograde des schémas ne permet pas de repartir de ceux identifiés lors de la précédente application. En effet, la toute dernière action jouée suffit à placer les nouveaux schémas avec le plus grand contexte dans des branches très éloignées des précédents.

5.3.1.2. Vers une sélection incrémentale des schémas

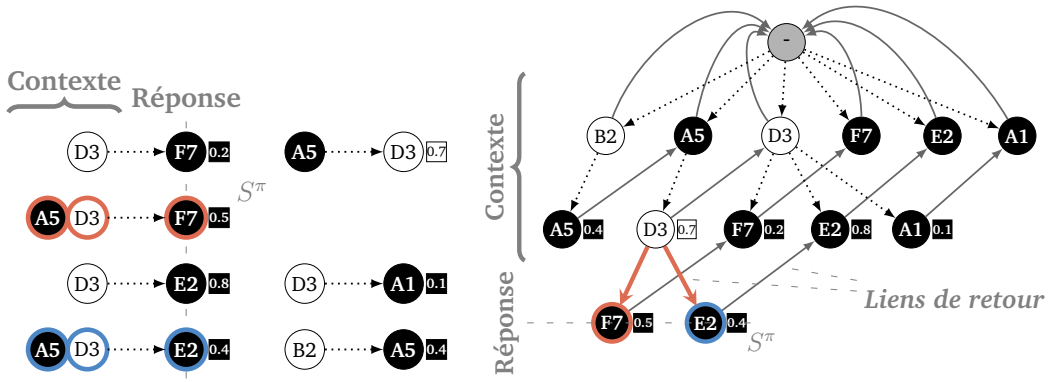
Pour chaque nouvel état atteint lors de la phase *roll-out*, la séquence des dernières actions jouées évolue d'une seule action : celle générée lors de l'itération précédente. Ainsi, d'un tour sur l'autre, le contexte des schémas applicables évolue d'une seule action ajoutée en suffixe du précédent contexte. En particulier, les plus grands contextes couverts par BHRF n'augmentent ainsi que d'une seule action au plus, mais peuvent se réduire de plusieurs actions d'un tour sur l'autre.

Une structure de données adéquate pour la politique de BHRF bénéficierait de cette évolution progressive des contextes afin d'identifier les nouveaux schémas en un temps constant d'un tour sur l'autre ($O(1)$). Les structures de données organisées suivant un sens chronologique comme les arbres de recherche sont propices à une évolution progressive dans l'espace des états. En outre, comme nous l'avons évoqué dans la section 5.2.3.2, cette étape dans le processus de sélection revient à simuler un arbre local de recherche. L'arbre direct de schémas formalise cet arbre local afin d'éviter de le reconstruire à chaque application de la politique BHRF.

5.3.2 Arbre direct de schémas

Un arbre direct de schémas organise des schémas temporels de tailles différentes suivant un ordre chronologique. Chaque noeud de l'arbre porte un schéma temporel et deux noeuds sont reliés entre eux si le schéma du noeud père (de taille n) correspond au contexte du schéma porté par le noeud fils (de taille $n + 1$). Les noeuds de cet arbre représentent à la fois l'application d'un schéma temporel et le contexte pour noeud suivant. Le noeud racine représente ici le contexte vide. Ainsi les schémas sont organisés par tailles croissantes comme autant de séquences d'actions possibles depuis un contexte vide, à l'instar d'un arbre de recherche classique (cf. définition 3.7 p.35).

À la différence d'un arbre de recherche classique, un noeud de l'arbre direct de schémas décrit plusieurs états de l'environnement. Il est ainsi nécessaire de pouvoir accéder rapidement depuis un noeud à l'ensemble des noeuds correspondants à des contextes applicables à un même état. En plus d'être reliés à des noeuds fils de taille supérieure, chaque schéma sera relié à un unique noeud de retour (*backtracking* en anglais) de taille inférieure tel que : un noeud portant un schéma (de taille n) est relié à un noeud de retour (de taille $n - 1$), si le schéma porté par ce dernier comporte une action en moins au préfixe de son contexte. La figure 5.6 illustre l'équivalence entre l'organisation des schémas temporels dans cette structure de données.



(a) Schémas temporels **(b)**Arbre direct de schémas

Figure 5.6.– Arbre direct de schéma

L'arbre direct de schémas se parcourt de deux façons différentes qu'il est possible de résumer ainsi, depuis un noeud quelconque :

— aller vers un noeud fils, revient à ajouter une action en suffixe de la séquence et considérer un schéma avec un plus grand contexte ;

— aller vers le noeud de retour, revient à supprimer l'action en préfixe de la séquence et considérer un schéma avec un plus petit contexte.

Cette structure de données ne correspond pas à un arbre suivant la théorie des graphes car les noeuds de retour induisent de nombreux cycles. Cependant pour sa similarité avec un arbre de recherche et par opposition à la forêt rétrograde de schémas, nous avons néanmoins choisi le terme d'arbre direct de schémas.

Dans la forêt rétrograde de schémas, nous supposions que les schémas sans contexte (la réponse seule) étaient préalablement construits : il s'agissait des racines de chacun des arbres. De la même manière nous considérons ici que ces mêmes schémas sont construits en tant que fils de la racine de l'arbre direct. Dans le cadre de jeux à deux joueurs, il est impératif de distinguer par quel joueur est joué chacune des actions. En l'occurrence, le noeud à la racine du pseudo arbre, sera le seul noeud à avoir des fils correspondant aux actions réalisables par les deux joueurs. Les autres noeuds sont comme les noeuds d'un arbre de recherche classique : le joueur ayant le trait alterne à chaque niveau.

5.3.3 Mise à jour de la représentation optimisée

L'arbre direct de schémas a été conçu pour organiser efficacement les schémas couverts par BHRF au travers notamment de la forêt rétrograde de schémas présentée en section 5.2.1.

Afin d'accumuler les mêmes connaissances que la forêt rétrograde de schémas, l'arbre direct devra effectuer les mêmes renforcements que ceux présentés dans la section 5.2.2 à la fin de chaque épisode. Ainsi l'arbre direct de schémas sera renforcé à partir de la même rétroaction que celle fournie à l'arbre de recherche de MCTS : la séquence d'action sélectionnée au cours de la phase de *descent* et la récompense obtenue à la fin de l'épisode. À l'instar de la forêt rétrograde, la mise à jour de l'arbre direct de schémas devra alors respecter les règles suivantes :

1. un schéma couvert par BHRF est renforcé, s'il apparaît dans la séquence des actions jouées dans la phase *descent* ;
2. un nouveau schéma est ajouté s'il apparaît dans la séquences d'action de la phase *descent* et s'il comporte une action de plus au préfixe d'un schéma déjà couvert par BHRF.

Nous présentons ici les algorithmes capables de mettre en oeuvre un tel renforcement pour un arbre direct de schémas. Dans la section 5.3.3.1 , nous revenons sur les propriétés autorisant à organiser les schémas de BHRF sous la forme d'un arbre direct de schémas. Dans la section 5.3.3.2 , nous présentons les modifications à apporter aux phases *update* et *growth* pour renforcer l'arbre direct de schémas. Enfin nous évoquons la complexité des algorithmes présentés à l'aune de ceux pour la forêt rétrograde de schémas, dans la section 5.3.3.3.

5.3.3.1. Équivalence entre les deux structures de données

L'agencement des schémas temporels dans l'arbre direct de schémas impose l'existence de certains schémas temporels. Grâce aux propriétés 5.1 et 5.2 , nous avons la certitude qu'il est possible de représenter les schémas de BHRF sous la forme d'un arbre direct de schémas. En effet la propriété 5.1 de la forêt rétrograde garantit que, pour tout schéma de BHRF, il existe un schéma correspondant à son noeud de retour dans l'arbre direct. De même, la propriété 5.2 de la mise à jour BHRF garantit que, pour tout schéma de BHRF, l'ensemble des schémas correspondants aux noeuds pères successifs jusqu'à la racine existent aussi.

5.3.3.2. Modifications apportées aux phases *update* et *growth*

Lors des phases de *update* et *growth*, l'arbre direct de schémas doit ajuster sa représentation comme la forêt rétrograde de schémas. Étant donné l'organisation chronologique des schémas dans l'arbre, il est préférable de parcourir l'arbre direct conjointement à la séquence des actions jouées lors de la phase de *descent*. Ce faisant nous pouvons nous positionner systématiquement sur le noeud correspondant au

plus grand contexte couvert par BHRF, pour la séquence de *descent* actuellement parcouru.

La mise à jour démarre de la racine pour le contexte vide et progresse par étapes dans l'arbre direct, comme cela est présenté dans l'algorithme 1 À chaque étape de la mise à jour, l'algorithme considère une nouvelle action de la séquence de *descent* suivant l'ordre chronologique.

Algorithm 1 Mise à jour (*update* et *growth*) - arbre direct de schémas

```
procedure UPDATESEQUENCE(descentSequence : Array<Move>, outcome : float)
    currentNode,nextNode : Node
    currentNode ← self.getRoot()
//Follow descent sequence
    for i ← 0 to descentSequence.size do
        nextNode ← currentNode.getChild(descentSequence[i])
//Create the node if it does not exist for the current one
        if nextNode == null then
            nextNode ← createNextNode(currentNode,descentSequence[i])
        end if
        currentNode ← nextNode
        if opponentMove(i)  then
            updateBackTrackingNodes(currentNode,inv(outcome))
        else
            updateBackTrackingNodes(currentNode,outcome)
        end if
    end for
end procedure
```

Si cette action est la réponse d'un schéma présent parmi les noeuds fils, l'arbre est descendu en suivant ce noeud. Comme la mise à jour ne considère qu'une action à la fois, le contexte maximal couvert par l'arbre augmente au plus d'une action à chaque étape. Le schéma atteint correspond alors au nouveau contexte maximal couvert par l'arbre direct, à ce stade de la séquence de *descent*.

Si aucune réponse parmi les noeuds fils ne correspond à la nouvelle action, cela signifie qu'il n'existe pas de schéma couvert par BHRF pour le contexte atteint : un nouveau noeud doit alors être ajouté. L'algorithme 2 détaille l'ajout d'un nouveau schéma à l'arbre direct depuis le noeud atteint. Cet algorithme remonte progressivement l'arbre par les noeuds de retour, jusqu'à atteindre le plus grand contexte pour lequel cette réponse peut être ajoutée. Il s'agit du premier contexte pour lequel il est possible de relier le nouveau schéma créé à un noeud de retour, c'est à dire à un schéma comportant exactement une action de moins au niveau du préfixe. Nous retrouvons ainsi la même règle de création de nouveaux schémas que celle de la forêt rétrograde. Avec cet algorithme par ailleurs, le noeud nouvellement créé

Algorithm 2 Création de noeud (*growth*) - arbre direct de schémas

```
procedure CREATENEXTNODE(fatherNode : Node, nextMove : Move) : Node
// fatherNode : node currently reached in the tree (Input and Output parameter)
    backNode : Node
// Check if the backtrackingnode of the new node exists
    backNode ← fatherNode.getBacktrackingNode()
    backNode ← backNode.getChild(Move)
    while backNode == null do
// Otherwise backtrack one more time
        fatherNode ← fatherNode.getBacktrackingNode()
        backNode ← fatherNode.getBacktrackingNode()
        backNode ← backNode.getChild(Move)
    end while
    return createNode(fatherNode,backNode, nextMove)
end procedure
```

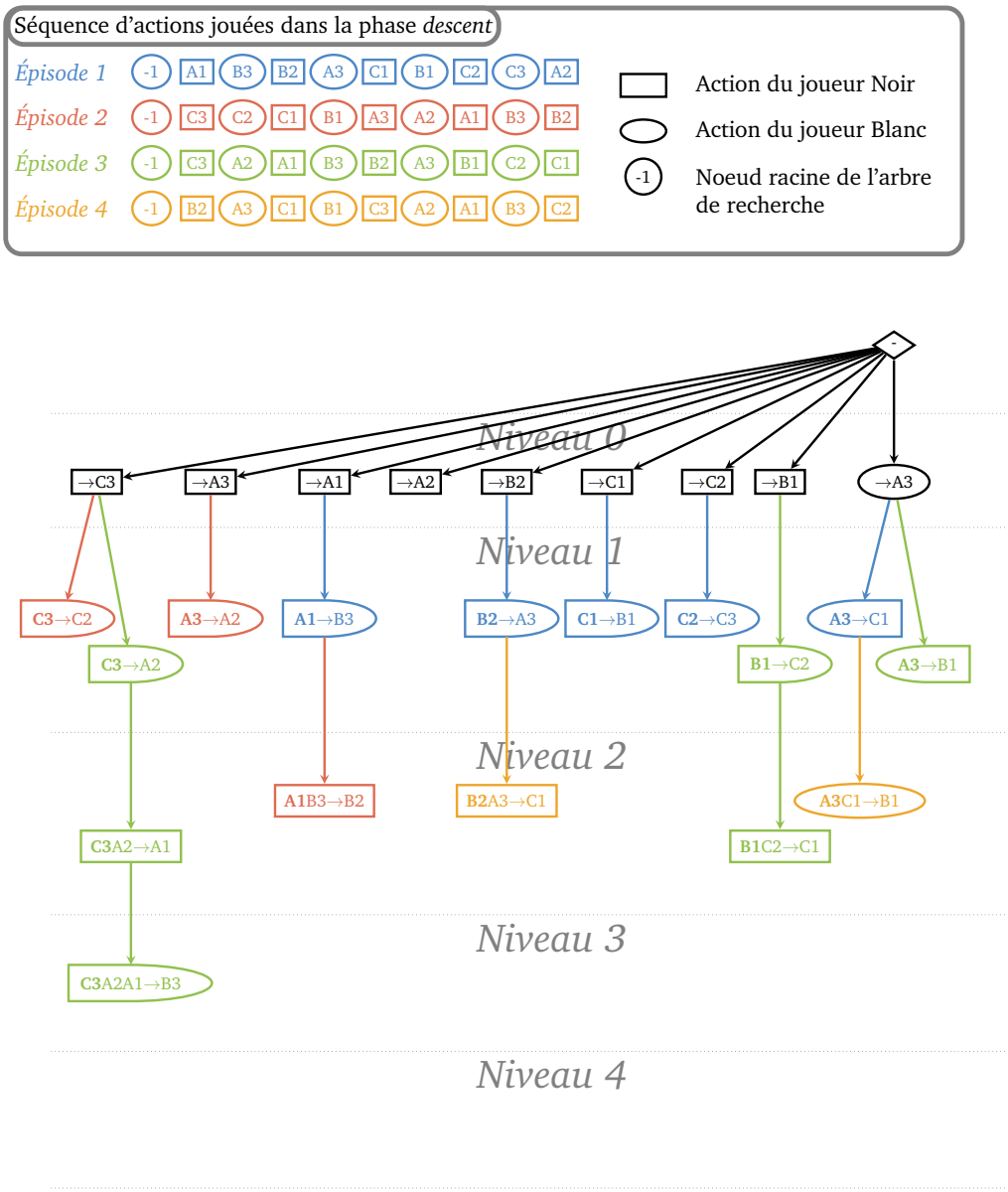
correspond, là encore, au nouveau contexte maximal couvert par l'arbre direct, à ce stade de la séquence de *descent*.

Algorithm 3 Mise à jour d'un noeud (*update*) - arbre direct de schémas

```
procedure UPDATEBACKTRACKINGNODES(startNode : Node, outcome : float)
    currentNode,backNode : Node
    currentNode ← startNode
    backNode ← currentNode.BackTrackingNode()
// Only the root node is its own backtracking node
    while currentNode != backNode do
// Update and backtrack
        currentNode.updateMean(outcome)
        currentNode ← backNode
        backNode ← currentNode.BackTrackingNode()
    end while
end procedure
```

À la fin de chaque étape, l'algorithme 3 renforce tous les schémas avec des contexte légèrement moins précis pour cette même réponse. Il suffit de remonter les noeuds de retour jusqu'à la racine, indépendamment de la progression générale de la mise à jour. Comme le noeud atteint à chaque étape de la mise à jour correspond au plus grand schéma couvert par BHRF, la mise à jour couvrira ainsi tous les schémas de BHRF apparaissant dans la séquence de *descent* . Nous retrouvons alors la même règle de renforcement que celle de la forêt rétrograde de schémas.

La figure 5.7 illustre la construction de la structure pour plusieurs mises à jour successives. L'arbre direct de schémas initialement vide est est progressivement étendu à la fin de chaque mise à jour suivant l'algorithme de construction proposé. Les séquences de la phase de *descent* proposées dans cet exemple sont exactement les mêmes que celles de la figure 5.4 pour la forêt de schémas rétrograde.


Séquence d'actions jouées dans la phase *descent*
Épisode 1
-1 A1 B3 B2 A3 C1 B1 C2 C3 A2
Épisode 2
-1 C3 C2 C1 B1 A3 A2 A1 B3 B2
Épisode 3
-1 C3 A2 A1 B3 B2 A3 B1 C2 C1
Épisode 4
-1 B2 A3 C1 B1 C3 A2 A1 B3 C2
Action du joueur Noir
Action du joueur Blanc
-1
Noeud racine de l'arbre de recherche
-
Niveau 0
→C3 →A3 →A1 →A2 →B2 →C1 →C2 →B1 →A3
Niveau 1
C3→C2 A3→A2 A1→B3 B2→A3 C1→B1 C2→C3 A3→C1
C3→A2 B1→C2 A3→B1
Niveau 2
A1B3→B2 B2A3→C1 A3C1→B1
C3A2→A1 B1C2→C1
Niveau 3
C3A2A1→B3
Niveau 4

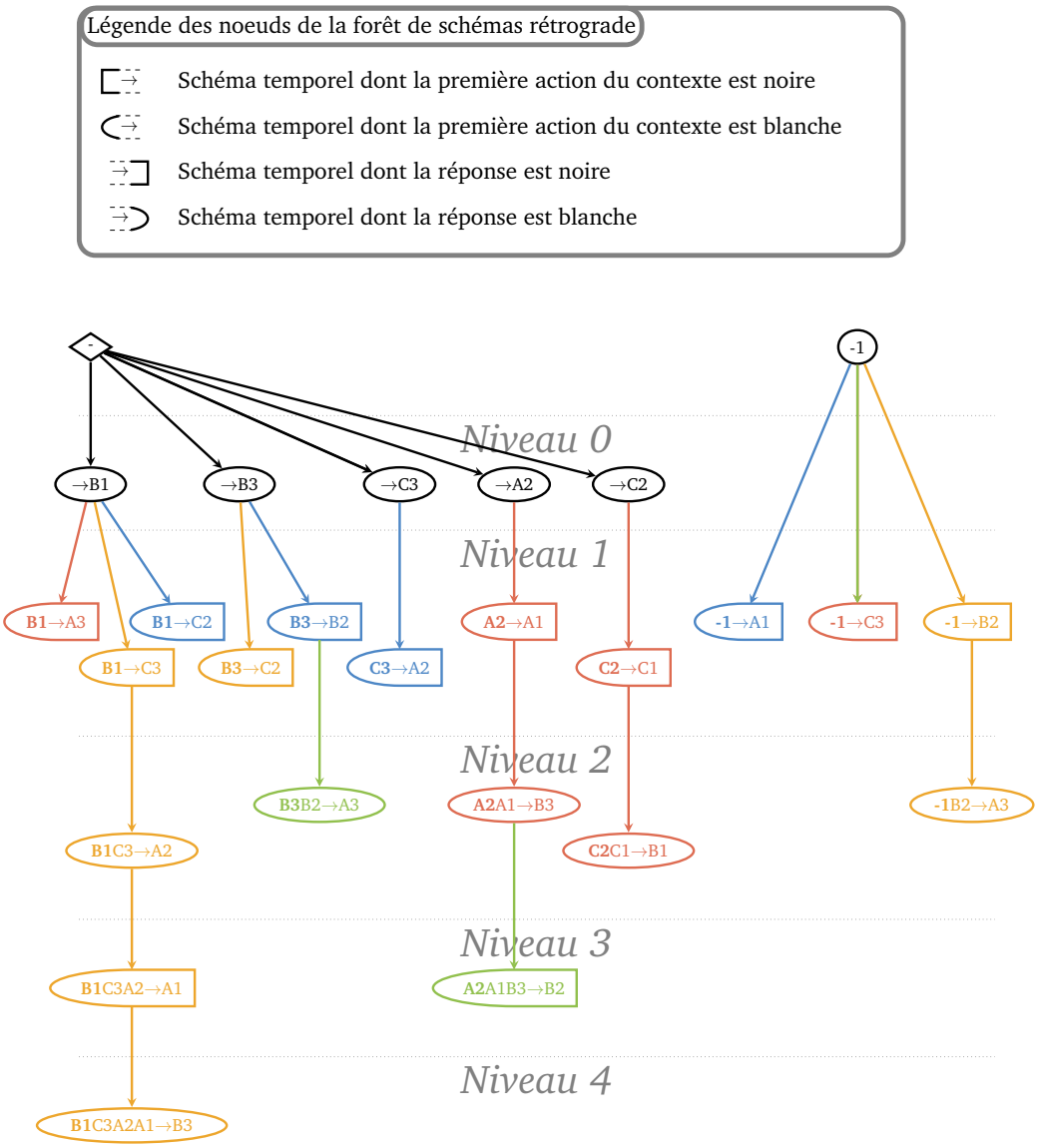


Figure 5.7.– Construction incrémentale d'un arbre direct de schémas

Comme ces deux structures de données sont équivalentes, les mêmes schémas temporels seront progressivement accumulés. Cependant les schémas seront organisés différemment suivant la structure choisie. Ainsi, sur une même branche de la forêt de schémas rétrograde (cf. figure 5.4), la couleur de la première action des schémas couverts (au niveau du contexte) alterne,à chaque changement de niveau, entre une action du joueur noir et une action du joueur blanc (la réponse reste identique). En revanche sur une branche de l'arbre direct de schémas, c'est la couleur de la réponse qui alterne à chaque changement de niveau à l'instar d'un arbre de recherche classique.

Par ailleurs, l'extension de l'arbre direct de schémas est bien moins intuitive à appréhender que celle de la forêt de schémas rétrograde. En effet, la forêt de schémas rétrograde ajoute au maximum un schéma par arbre au cours d'une même mise à jour. À l'inverse dans l'arbre direct de schémas, plusieurs schémas temporels sont susceptibles d'être créés sur une même branche. Par exemple, l'épisode 4 dans la figure 5.7 introduit quatre nouveaux schémas depuis le noeud $\rightarrow$B1.

5.3.3.3. Complexité de la mise à jour

La complexité de la mise à jour est semblable à celle de la forêt rétrograde de schémas. À chaque étape de la mise à jour, le traitement le plus coûteux est le renforcement de tous les schémas associés au contexte atteint (algorithme 3). Sa complexité est proportionnelle à la profondeur de l'arbre atteinte. Ainsi le pire des cas possibles survient lorsqu'aucun schéma n'a pas besoin d'être ajouté tout au long de la mise à jour. Dès qu'il est nécessaire de créer un nouveau schéma, la complexité pour sa création réduit d'autant la profondeur atteinte dans l'arbre. Dans le cas contraire, la profondeur croît d'une action à chaque étape. La complexité totale de la mise à jour est alors d'un ordre de grandeur $O(d^2)$, où d correspond à la taille de la séquence de *descent*. L'algorithme de mise à jour proposé ne dépend que du renforcement de l'arbre de recherche de MCTS et non de la forêt rétrograde de schémas. Cette nouvelle structure suffit à accumuler les schémas temporels de BHRF.

5.3.4 Politique de *roll-out* optimisée avec BHRF

Comme nous l'avons évoqué dans la section 5.3.1, l'arbre direct de schémas a été conçu afin de réduire le temps de calcul nécessaire à la politique de BHRF. En particulier, la sélection des schémas avec le plus grand contexte représente une étape très coûteuse pour une structure de données comme la forêt rétrograde de schémas.

L'organisation proposée par l'arbre direct de schémas facilite l'identification de ces schémas, d'un état sur l'autre. À cet effet, l'arbre direct de schémas est parcouru conjointement à l'avancement de l'épisode, de manière à se positionner systématiquement sur le noeud correspondant au plus grand schéma applicable par BHRF. Ainsi, à chaque application de la politique BHRF, le système a accès en un temps réduit aux schémas désirés, pour la situation courante.

Dans la section 5.3.4.1, nous précisons le parcours à réaliser dans l'arbre direct suite à chacune des actions jouées au cours de l'épisode (phases *descent* et *roll-out*). Dans la section 5.3.4.2, nous évoquons les modifications à apporter au processus de sélection de réponses pour la politique de BHRF. Enfin la section 5.3.4.3 présente la complexité atteinte pour la sélection des schémas de plus grand contexte.

5.3.4.1. Progression dans l'arbre direct de schéma

La progression dans l'arbre direct de schémas est réalisée afin de se positionner sur le noeud correspondant au schéma applicable de la plus grande taille, pour chacun des états traversés au cours de l'épisode. Le parcours de l'arbre au cours de l'épisode est similaire à celui réalisé lors de la mise à jour à partir des actions jouées lors de la phase *descent* (cf. section 5.3.3.2). Pour chaque action jouée, l'algorithme 4 progresse à partir de la position précédente suivant les deux règles suivantes :

— Si un schéma parmi les noeuds fils possède la même réponse que l'action jouée, ce noeud représente la nouvelle position atteinte ;

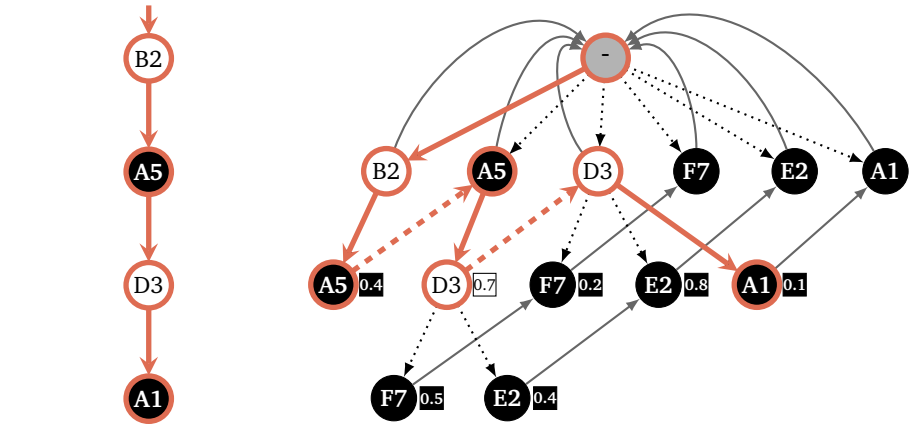
— Si aucun noeud fils correspond à la réponse jouée, l'arbre direct est successivement remonté par les noeuds de retour jusqu'à ce que la précédente règle soit vérifiée.

Algorithm 4 Parcours de l'arbre direct de schémas

```
procedure PROGRESS(startNode : Node, nextMove : Move ) : Node
    currentNode,nextNode : Node
    currentNode ← startNode
    nextNode ← currentNode.getChild(nextMove)
// Find a node with the next move
    while nextNode == null do
        currentNode ← currentNode.BackTrackingNode()
        nextNode ← currentNode.getChild(nextMove)
    end while
// Move to that node
    currentNode ← nextNode
end procedure
```

Dans la figure 5.8, nous proposons une illustration de ces deux règles pour l'application successive de ce même algorithme à une séquence d'actions. La première

règle sert à suivre l'avancement de la partie au regard de la dernière action jouée. La deuxième règle est appliquée lorsque la taille du contexte atteint est trop précise pour les schémas actuellement couverts par BHRF. Il convient de réduire progressivement cette taille jusqu'à pouvoir atteindre un schéma correspondant à cette réponse. Dans le pire des cas, cette réponse existe toujours au niveau de la racine (pour le contexte vide). Par ailleurs, la profondeur atteinte dans l'arbre direct n'augmente au maximum que d'une action par action jouée mais peut réduire, au maximum, d'autant de niveaux que ceux atteints jusqu'alors.



(a)Séquence d'actions jouées **(b)**Arbre direct de schémas

Figure 5.8.– Progression dans l'arbre direct de schémas en suivant les plus grands contextes.

5.3.4.2. Processus optimisé de sélection de réponse

Comme nous l'avons présenté dans la section 5.2.3, le processus de sélection des actions suivant la politique BHRF se déroule en trois étapes. L'optimisation introduite par l'arbre direct de schémas porte exclusivement sur la première étape : la sélection déterministe des schémas. Les deux autres étapes du processus de sélection restent rigoureusement identiques. Le déroulement complet de la politique de *roll-out* est résumé dans l'algorithme 5. Nous en resterons ici à décrire les modifications apportées par rapport à l'algorithme de la politique de *roll-out* pour une forêt rétrograde de schémas consultable en Annexe A.5.

5.3.4.3. Complexité de la sélection des schémas pour l'arbre direct

Pour le calcul de la complexité, nous nous plaçons à nouveau dans le pire des cas. En l'occurrence, nous considérons que la politique BHRF est appliquée systématiquement à chacune des étapes de la phase *roll-out* ($\epsilon = 100$). Dans la section 5.3.1, nous avions estimé le coût pour la sélection des schémas dans la forêt rétrograde à un

ordre de grandeur de $0(b^2r)$ (dans le pire des cas), avec r le nombre d'état traversé dans la phase de *roll-out* et b le nombre moyen d'actions légales à chaque étape. L'utilisation de l'arbre direct de schémas simplifie l'identification de ces schémas au prix de deux traitements supplémentaires : le parcours de l'arbre direct tout au long de l'épisode, présenté dans la section 5.3.4.1, et l'identification des réponses légales lorsque la politique BHRF est appliquée, présentée dans la section 5.3.4.2.

Bien que le parcours de l'arbre direct soient réalisé au cours des phases *descent* et *roll-out*, sa complexité reste négligeable au regard des autres traitements évoqués dans cette section. Comme nous l'avons évoqué dans la section 5.3.4.1, l'arbre direct est descendu d'un niveau au maximum par état traversé et est remonté d'un nombre de niveaux égal au maximum à la profondeur atteinte jusqu'alors. Ainsi tout approfondissement réalisé au cours de l'épisode entraîne au maximum un nombre équivalent de remontées. La complexité de cet algorithme est environ égale à $O(2r+2d))$ avec r et d les nombres d'états traversés respectivement dans les phases de *roll-out* et *descent*.

L'identification des réponses légales peut s'avérer relativement coûteuse pour une application isolée de la politique BHRF. En effet, cette vérification requiert de parcourir tous les noeuds fils afin de s'assurer que leurs réponses sont applicables à la situation courante. Dans le cas contraire, cette opération est répétée pour des contextes de plus en plus petits jusqu'à éventuellement atteindre la racine. La complexité de ce traitement pour une application isolée s'élève à un ordre de grandeur de $O(cp)$ dans le pire des cas, avec c le nombre moyen de noeuds fils dans l'arbre direct de schémas et p la profondeur atteinte avant l'application de la politique. Cependant, au niveau d'un épisode, le nombre total de remontées dans l'arbre direct de schémas est limité par le nombre total de descentes réalisées avec l'algorithme de progression. Ainsi la complexité au niveau de l'épisode est bornée par $O(2rc + dc))$. Par cette nouvelle structure de données, nous passons d'un ordre de grandeur quadratique suivant le nombre moyen d'actions légales à un ordre de grandeur linéaire suivant le nombre moyen de noeuds fils dans l'arbre direct [3].

3. Nous faisons ici l'hypothèse que la valeur c est en général inférieure à b. Il existe cependant des situations pour lesquelles il est plus désavantageux de considérer les réponses couvertes parmi les noeuds fils plutôt que les réponses légales. Par exemple lorsque le nombre d'actions légales est petit et le nombre de noeuds fils illégaux important.

Algorithm 5 Politique de *roll-out* incrémentale pour BHRF

```
procedure ROLLOUTPOLICY(lastMove : Move, currentNode : Node ) : Move
//lastMove : move previously selected
//currentNode : current position in the BHRF forward tree
//(initialized according to the descent sequence)
    candidate : Move
    lstCandidate : List<Move>
// Progress through the tree
    currentNode ← progress(currentNode,lastMove)
//Probability ϵ to use BHRF
    if randomValue() < ϵ then
// Get legal childs
        for candidate : Node ∈ currentNode.Childs() do
            if IsLegal(candidate) then
                lstCandidate.add(candidate)
            end if
        end for
// If no legal child, backtrack and repeat
        while lstCandidate.empty() do
            currentNode ← currentNode.BackTrackingNode()
            for candidate : Node ∈ currentNode.Childs() do
                if IsLegal(candidate) then
                    lstCandidate.add(candidate)
                end if
            end for
        end while
// Selects the reply according to Equation (5.1)
        candidate = CandidateSoftMaxUct(lstCandidate)
        if randomValue() < candidate.uctEstimate() then
            return candidate
        end if
    end if
// Plays default policy otherwise
    return defaultRandomMove()
end procedure
```

5.4 Évaluation volatile des représentations

À l'instar des états couverts par l'arbre de recherche, nous avons choisi d'évaluer les schémas accumulés par BHRF. Ces évaluations servent à pondérer le choix des différentes réponses candidates lors de la politique de *roll-out* (cf. section 5.2.3). Comme nous l'avons vu dans la section 5.2.3.3, le calcul de la probabilité associée à chacune des réponses candidates peut s'avérer coûteux en temps de calcul. Dans cette partie, nous proposons de simplifier à l'extrême les évaluations associées aux schémas temporels afin de faciliter la sélection des réponses lors de la politique BHRF : à chaque contexte correspond alors une unique réponse possible.

Dans la section 5.4.1, nous revenons sur les motivations d'une telle simplification. Nous montrerons notamment comment le choix effectué par la politique de BHRF est alors délégué à la politique de *descent*. Dans la section 5.4.2, nous précisons les modifications à apporter au renforcement de la structure de données complémentaire. Enfin la section 5.4.3 détaille la nouvelle politique de BHRF avec des évaluations volatiles. Nous considérerons ici que les schémas temporels sont accumulés dans un forêt rétrograde de schémas (cf. section 5.2.1), afin de mettre en avant les différences avec le modèle initial de BHRF. Cependant cette politique proposée se prête mieux à la structure de données optimisée, présentée dans la partie précédente (cf. section 5.3) [4].

5.4.1 Motivation pour des évaluations volatiles

Dans la méthode MCTS, les évaluations accumulées dans l'arbre de recherche sont sujettes à deux interprétations liées. D'une part, ces évaluations correspondent à une approximation de la probabilité de gain depuis cet état, dans l'absolu. D'autre part, ces évaluations sont relatives aux différentes expériences vécues par l'agent jusqu'à présent. Dans la section 5.4.1.1, nous revenons plus longuement sur cette ambivalence dans les évaluations apprises.

Les méthodes d'apprentissage par renforcement disposent de mécanismes afin de réguler l'importance des dernières expériences sur l'évaluation. Dans la section 5.4.1.2, nous développons cet aspect des méthodes d'apprentissage par renforcement. En particulier, nous présentons les évaluations volatiles ne tenant compte que de la toute dernière expérience. Enfin, nous exposons dans la section 5.4.1.3 comment nous souhaitons utiliser des évaluations volatiles pour BHRF afin de simplifier la sélection des réponses pour un même contexte.

4. Les algorithmes détaillés présentés en Annexes A.6 et A.5 utiliseront un arbre direct de schémas

5.4.1.1. Évaluations relatives aux expériences

Les évaluations apprises par renforcements s'inscrivent systématiquement dans une dynamique régie par l'itération de la politique généralisée (cf. Définition 3.17 p.52). Au fur et à mesure des épisodes, les évaluations des représentations fournissent tout d'abord une approximation grossière, pour ensuite progressivement converger vers la valeur absolue de cette représentation. C'est par le grand nombre d'expériences différentes que l'agent apprend à évaluer une représentations indépendamment de tout contexte. Cependant lorsque l'apprentissage est interrompu avant son terme, les évaluations obtenues sont relatives aux expériences vécues jusqu'à présent par l'agent. Ainsi les évaluations obtenues seront différentes d'une application à l'autre de la même méthode selon les expériences vécues.

Dans le cadre de jeux combinatoires, les méthodes d'apprentissage par renforcement ont rarement le temps d'être appliquées jusqu'à leur convergence [5]. Ainsi les évaluations apprises sont relatives aux connaissances accumulées par l'agent jusqu'à présent et restituent une évolution possible de leurs valeurs. Par exemple pour estimer le même état, deux applications indépendantes de MCTS sont susceptibles de fournir des évaluations différentes suivant les états qui ont été explorés au cours des apprentissages respectifs. En pratique, les évaluations d'un même état sont similaires mais il existe des situations pour lesquelles une exploration chanceuse permet d'identifier des pièges difficiles à déceler [RS11].

Ce relativisme est d'autant plus exacerbé que les représentations sont initialement biaisées, comme les schémas temporels par exemple. À la différence d'un état couvert par l'arbre de recherche, ces représentations ne vérifient plus la propriété de Markov. Par conséquent les actions réalisées avant le contexte du schémas auront une influence décisive sur l'évaluation résultante, en plus des actions sélectionnées ensuite. En particulier pour BHRF, les schémas renforcés à la fin de chaque épisode dépendent des actions sélectionnées par la politique $\pi_{descent}$. Ainsi, l'évolution spécifique de cette politique pour une situation donnée imprègne aussi les évaluations accumulées par BHRF, qui évoluent en conséquence.

5.4.1.2. Contrôle de l'évolution des évaluations

Dans les méthodes d'apprentissages par renforcement, l'ajustement successif des évaluations est régulé par le taux d'apprentissage (cf. paragraphe Évaluation de la politique p. 54 dans la section 3.2.3.4)). En principe, ce taux décroît progressivement

5. À titre d'exemple, la convergence de la méthode de Monte Carlo est garantie lorsque tous les états de l'environnement ont été visités un nombre infini de fois [SB98] : ce qui est bien évidemment hors de question pour les jeux combinatoires où le nombre d'états en lui-même est déjà très grand.

suite aux renforcements afin que les évaluations se stabilisent. C'est notamment le cas dans la méthode MCTS à l'instar de n'importe quelle méthode de Monte Carlo (cf. section 4.1.2.4). Lorsque ce taux a une valeur constante, les dernières expériences vécues ont une plus grande influence sur l'évaluation courante de la représentation. Ce faisant, les évaluations apprises sont encore plus influencées par le contexte courant de l'apprentissage [6].

Suivant cette même perspective, l'extrême inverse à un taux d'apprentissage décroissant consiste à ne tenir compte que de la dernière expérience réalisée pour évaluer une représentation. Les évaluations des représentations forment une mémoire à très court terme susceptible de changer à la fin de chaque épisode. Ces évaluations suivent au plus près le contexte courant d'apprentissage au risque de ne jamais se stabiliser. Nous parlerons alors d'évaluations volatiles (ce terme a été choisi ici par analogie avec un gaz ou une phéromone).

À titre d'exemple, les évaluations des schémas temporels dans l'amélioration LGRF[Dra09 ; BD10] sont volatiles (section 4.2.2.4) car d'un épisode à l'autre, la réponse associée à un même contexte est susceptible de radicalement changer. D'après Drake et Baier, un tel renforcement permettrait de suivre au plus près l'évolution de la partie. Cependant pour BHRF, une évaluation volatile permettra de suivre au plus près les évolutions de la politique $\pi_{descent}$.

5.4.1.3. Évaluations volatiles pour BHRF

Afin d'« extraire » des connaissances présentes dans l'arbre de recherche, ces évaluations sont calculées à partir des mêmes rétroactions. Ainsi les schémas de BHRF sont renforcés à partir des actions sélectionnées par une autre politique : la politique de *descent* [7]. Dans une certaine mesure, la politique $\pi_{descent}$ choisit déjà la réponse à apporter aux différents contextes rencontrés lors de la phase *descent*. Là aussi, ce choix s'effectue à partir des différentes évaluations accumulées dans l'arbre de recherche. Cependant ces évaluations correspondent à des représentations bien plus précises que des schémas partiels. Ainsi l'objectif de cette variante de BHRF consiste à faire reposer le choix des réponses dans de la politique de *roll-out* sur celui ayant été effectué par la politique $\pi_{descent}$.

Au fur et à mesure des épisodes, les évaluations dans l'arbre de recherche seront affinées et, par conséquent, les réponses choisies par la politique $\pi_{descent}$ seront de

6. Le taux d'apprentissage est généralement laissé constant lorsque l'environnement est susceptible d'évoluer de lui-même[SB98]

7. En apprentissage par renforcement, de telles méthodes sont qualifiées *off-policy* ; à la différence des méthodes *on-policy* où les représentations sélectionnées par la politique seront celles renforcées (comme la politique de *descent* de MCTS).

plus en plus pertinentes. Afin de suivre au plus près les évolutions de la politique $\pi_{descent}$, nous utilisons ici des évaluations volatiles pour les schémas temporels de BHRF. À l'instar de la méthode LGRF, une seule réponse sera associée à chaque contexte différent. Cette réponse sera susceptible de changer radicalement d'un épisode à l'autre. Cependant, à la différence de LGRF, nous ne tiendrons même plus compte du résultat de la simulation. Les réponses associées à chaque contexte de BHRF correspondent à celles sélectionnées par la politique $\pi_{descent}$ la dernière fois que ce contexte est apparu lors de la phase *descent*. Nous faisons ici l'hypothèse qu'il s'agissait alors de la meilleure réponse pour la politique $\pi_{descent}$ au regard des expériences vécues jusqu'alors. Ainsi les schémas temporels accumulés par BHRF n'ont plus besoin d'être évalués, il suffit de mémoriser la dernière réponse sélectionnée pour chaque contexte couvert.

5.4.2 Mise à jour volatile de la représentation

La mise à jour de BHRF recouvre aussi bien le renforcement des schémas couverts par la représentation complémentaire (phase de *update*) que l'ajout incrémental de nouveaux schémas (phase de *growth*). L'amélioration proposée dans cette partie ne concerne que le renforcement des évaluations des schémas. La procédure d'ajout de nouveaux schémas reste rigoureusement la même à celle présentée dans la section 5.2.2.1 : un schéma est ajouté s'il comporte une action de plus au niveau du préfixe d'un schéma déjà couvert.

Au cours de la phase *update*, un schéma couvert par BHRF est renforcé, s'il apparaît dans la séquence d'actions jouées lors de la phase *descent*. Dans le cadre d'évaluations volatiles, cette même rétroaction est interprétée différemment : elle ne sert pas à ajuster l'évaluation de chacun des schémas. Les schémas renforcés définissent, en l'occurrence, la nouvelle réponse à appliquer à chacun des contextes associés [8]. Ainsi, si le même contexte survient lors d'une prochaine phase de *roll-out*, la réponse appliquée correspond à celle du dernier schéma à avoir été renforcé pour ce contexte. Conformément à la définition d'une politique (Définition 3.15 p. 50) , la politique de BHRF est alors établie au moment de mise à jour de la représentation, simplifiant de fait sa mise en application.

Les algorithmes de mise à jour pour des évaluations volatiles sont consultables en Annexe A.6, pour une représentation complémentaire sous la forme d'un arbre direct de schémas.

8. Si le même contexte apparaît plusieurs fois de suite dans la séquence de *descent*, la réponse jouée le plus tard au cours de la séquence sera mémorisée.

5.4.3 Politique de *roll-out* avec des évaluations volatiles

La politique de BHRF vient compléter la politique de *roll-out* à partir des connaissances accumulées dans la structure de donnée complémentaire. Les actions du *roll-out* sont alors sélectionnées à partir des réponses préconisées par les schémas de BHRF. À l'instar de la politique présentée dans la section 5.2.3.1, la réponse adéquate est sélectionnée suivant un processus de plusieurs étapes successives, exécuté à chaque application de la politique de BHRF. Cependant ce processus est ici considérablement simplifié. Nous pouvons distinguer deux étapes :

1. Sélection déterministe du contexte ;
2. Application déterministe de la réponse.

À nouveau l'application de la politique BHRF est régulée par une probabilité ϵ fixée *a priori*. Lorsque cette politique n'est pas appliquée ou suggère une réponse non légale, la politique de *roll-out* est ensuite appliquée. La proportion effective d'actions générées par BHRF est là encore strictement inférieure à ϵ puisque la politique BHRF est susceptible de suggérer des réponses non légales.

Le déroulement général du processus de sélection est résumé dans la figure 5.9 pour une forêt rétrograde de schémas. L'algorithme de la politique de *roll-out* décrite dans cette section est consultable dans l'annexe A.5 pour un arbre direct de schémas (bien plus efficace en terme de complexité algorithmique). Dans les paragraphes suivant nous allons revenir plus en détails sur les étapes de ce processus.

Sélection déterministe du contexte Dans un premier temps, le plus grand contexte couvert par les schémas de BHRF est identifiée pour la situation actuelle. En d'autres termes, il suffit d'identifier les schémas pour lesquels le contexte coïncide avec le plus grand nombre d'actions précédemment jouées.

Application déterministe de la réponse Ensuite la réponse associée à ce contexte est appliquée. Il s'agit en l'occurrence, de la dernière réponse sélectionnée par la politique de *descent* pour ce même contexte. Il est cependant possible que cette réponse soit illégale pour la situation courante du jeu [9]. Le cas échéant, aucune action n'est proposée par la politique BHRF ; la politique de *roll-out* par défaut sera alors appliquée à sa suite.

9. Nous avons réalisé ce choix afin de réduire au maximum le temps de calcul nécessaire pour la politique BHRF avec un arbre direct de schémas. Un tel choix n'optimise pas le temps de calcul pour une forêt rétrograde de schémas.

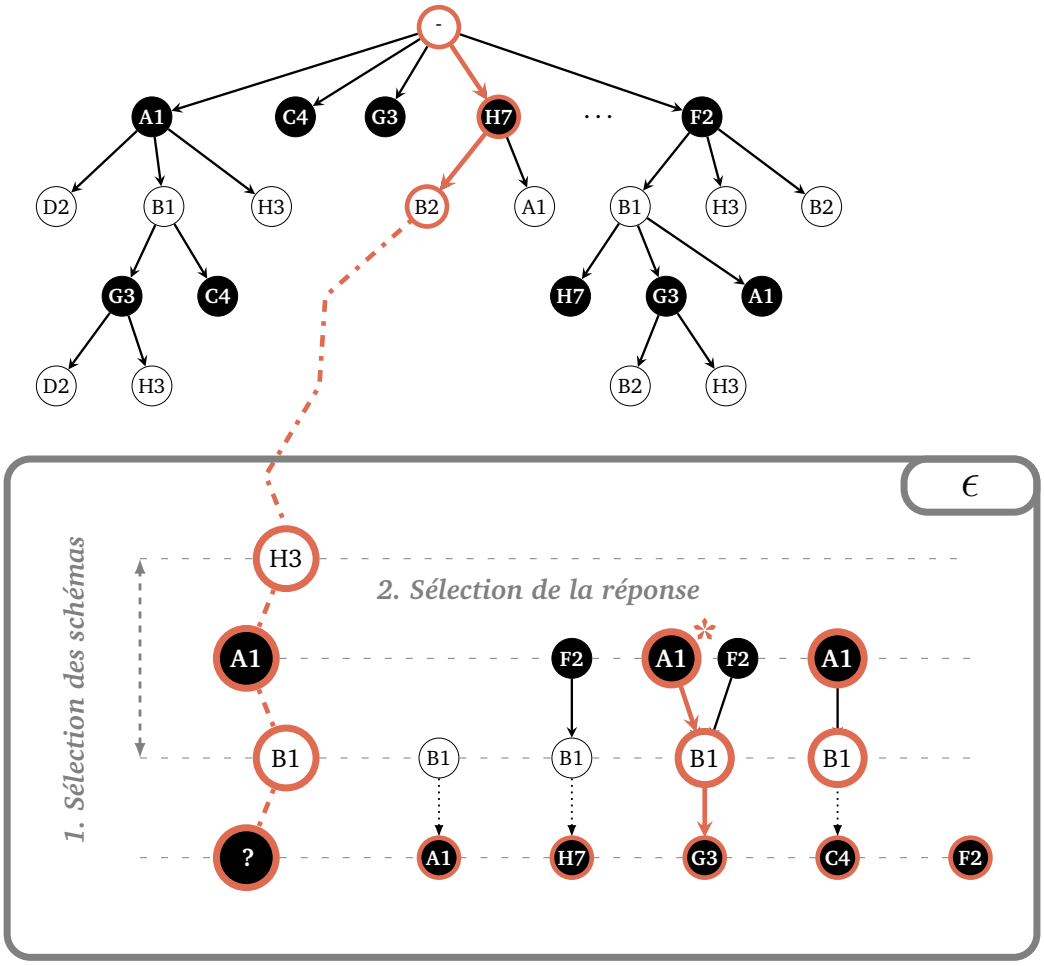


Figure 5.9.– Processus de sélection d'action de BHRF dans sa version volatile

5.5 Résultats expérimentaux

BHRF vise à améliorer les capacités d'apprentissage de MCTS. Dans cette section, nous analysons l'influence de BHRF dans le cadre du jeu de Go. Nous détaillerons dans un premier temps le protocole expérimental mis en oeuvre pour juger d'une telle influence dans la section 5.5.1. Les études réalisées porteront essentiellement sur la version non-volatile de BHRF, dans un second temps. La section 5.5.2 relate des études préliminaires sur le potentiel d'une telle approche sur une version limitée d'un programme professionnel. En particulier nous nous intéresserons à une étude qualitative des connaissances apprises par BHRF dans la section 5.5.3. Ensuite nous observerons l'influence de la version non-volatile de BHRF sur le même programme disposant de l'ensemble de ses capacités dans la section 5.5.4. Dans un troisième temps, nous étudierons l'influence de mise à jour volatile de BHRF sur le même programme au maximum de ses capacités dans la section 5.5.5.

5.5.1 Protocole expérimental

Nous avons choisi de développer BHRF pour le jeu de Go avec le programme open-source *Fuego* (Version 1.1) [Enz+09]. Ce programme implémente MCTS complété des principales améliorations développées pour le jeu de Go (cf. section 4.2.1) : UCT, RAVE, *prior knowledge* et politique de *roll-out sequence-like* (schémas spatiaux 3x3, schémas experts et proximité avec le dernier coup joué). Le programme *Fuego* par défaut obtient de très bonnes performances en 9x9 comme tous les programmes professionnels actuels. Les principales avancées à présent sont attendues pour un goban de 19x19.

Pour toute les expérimentations proposées, nous avons confronté le programme *Fuego* avec BHRF au même programme sans BHRF, appelé programme de référence. Le paramétrage est identique pour les deux programmes lorsque cela n'est pas explicitement mentionné. En particulier, nous allons considérer les paramètres suivants dans cette section :

— La taille du plateau (△) : `9x9,19x19` : détermine la difficulté du jeu. L'espace de recherche est bien plus large en 19x19. Le programme doit encore plus se concentrer sur les états les plus intéressants. En outre, les jeux réalisés sur des plateaux plus larges génèrent des situations encore plus complexes, qui ne se traitent pas facilement avec des connaissances expertes ;

— Le nombre de simulations (▼) : `1k, ..., 10k, 30k` : indique le nombre maximum d'épisodes autorisés à chaque tour. Un plus grand nombre de simulations implique des évaluations plus fines des état couverts par l'arbre de recherche et par conséquent, une politique de *descent* plus efficace ;

— Politique de *roll-out* experte (■) : `True, False` : indique si la politique de *roll-out* fait usage de représentations expertes comme dans une politique *sequence-like* ou bien est purement aléatoire ;

— BHRF : ϵ `= 0 .. 100%` (□) : contraint l'utilisation des représentations lors de la phase de *roll-out* (cf. section 5.2.3).

— Profondeur Maximale : `2,5,`∞ (⧫) : Limite la taille maximale du contexte des schémas couverts par BHRF (par défaut aucune limitation n'est fixée : ∞) ;

— Volatile : `True,False` (◎) : indique si BHRF dispose des évaluations volatiles présentées dans la section 5.4).

Afin de comparer les performances des programmes, nous utiliserons principalement le nombre moyen de victoire ou taux de victoires. Les taux de victoires sont le résultat moyen de nombreuses parties successives entre les deux programmes en alternant les couleurs. Comme le nombre de parties est variable suivant les expérimentations, l'ensemble des taux de victoires sont donnés avec un intervalle de confiance de 95%.

Les travaux présentés ici servent essentiellement à mettre en lumière l'intérêt d'une telle approche et non de produire un programme compétitif. C'est pourquoi les deux programmes disposent du même nombre fixe de simulations et ne jouent pas avec un temps limité, comme c'est le cas lors de compétitions. Comme nous l'avons évoqué précédemment la mise à jour de BHRF et son utilisation sont coûteuses en temps d'exécution. Sur la machine locale dont nous disposions (Intel(R) Core(TM) i7-2600 CPU 3.40 GHz avec 8GB de mémoire vive), nous constations une exécution ralentie d'un facteur allant de 4 à 12 suivant les tailles de plateau et le nombre de simulations considéré, pour le modèle BHRF présenté en section 5.2. Les améliorations de BHRF présentées dans les section 5.3 (arbre direct de schémas) et 5.4 (évaluation volatile) ont permis d'atteindre jusqu'à une exécution deux fois plus lente. Néanmoins l'implémentation réalisée reste encore trop lente pour rivaliser à temps limité avec les programmes actuels. Cependant à nombre de simulation fixé, la qualité des connaissances apprises est bien meilleure.

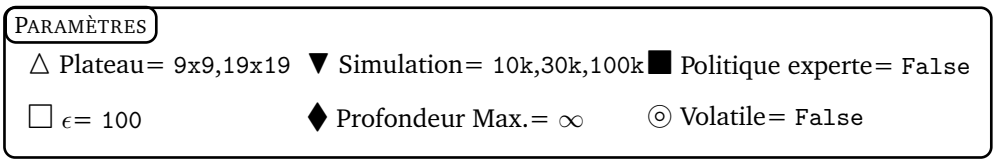
5.5.2 Politique de *roll-out* aléatoire avec BHRF

Les connaissances présentes dans BHRF sont acquises à la volée et donc différentes à chaque partie ; À la différence des connaissances expertes présentes dans les politiques *sequence-like*. Dans cette section, nous observons dans quelle mesure l'introduction de connaissances acquises à la volée par le programme bénéficient au programme.

Pour ces expériences, les politiques de *roll-out* expertes des deux programmes ont été désactivées. C'est à dire que la politique de *roll-out* par défaut est purement aléatoire. Les seules informations expertes spécifiques au jeu mobilisées, sont celles utilisées pour initialiser les noeuds de l'arbre. Dans la prochaine section nous étudierons l'influence de BHRF lorsque les deux compétiteurs disposent du même paramétrage commun. Par la suite , nous considérerons des expériences où le programme avec BHRF est désavantagé par rapport au programme de référence afin de mieux se rendre compte du comportement global du programme. Enfin nous observerons l'influence de la profondeur maximale sur les performances de BHRF dans une telle configuration.

5.5.2.1. Configuration équitable

Dans la table 5.1, nous indiquons le gain obtenu par le programme avec BHRF sur son adversaire [10] pour des plateaux de tailles 9x9 et 19x19 ainsi que différents nombres d'épisodes.



Simulations	10000	30000	100000
goban 9x9	$+17.3\% \pm 1.6$	$+16.9\% \pm 2$	$+18\% \pm 2.6$
goban 19x19	$+24.3\% \pm 3.5$	$+26.6\% \pm 2.5$	$+27.7\% \pm 3.4$

Table 5.1.– Victoires d'une politique de *roll-out* avec BHRF contre une purement aléatoire.

Ces premiers résultats nous montrent clairement que la réutilisation de connaissances provenant de l'arbre de recherche peuvent améliorer les performances globales du système. Le programme avec BHRF est bien meilleur que son adversaire pour chacune des configurations proposées. L'heuristique Pool-RAVE montrait des résultats semblables pour le jeu Go en 9x9 [11] sans utiliser de connaissances expertes lors de la phase de *roll-out*.

Les performances obtenues avec un goban 19x19 sont bien meilleures qu'avec un goban 9x9. Les parties sur un plateau 19x19 sont bien plus longues qu'en 9x9. De

10. Les résultats présentés indiquent le gain de performance obtenu par l'ajout de BHRF au programme de référence. Ainsi une valeur de $+27.7\%$ signifie que BHRF n'apporte un accroissement de performance de 27.7 par rapport au taux de victoire entre deux programmes de référence (c.a.d 50%) ; soit un taux de victoire total de 77.7%.

11. Les résultats en 19x19 ne sont pas fournis pour l'heuristique Pool-RAVE.

fait, les épisodes sont plus longs et les phases de *roll-out* en conséquence. Ainsi, les représentations stockées dans BHRF ont plus de chance d'avoir été sollicitées dans les *roll-out* de cette taille. En outre les parties en 19x19 disposent d'un bien plus grand espace de recherche. Une meilleure sélection des actions a un plus grand impact sur les performances globales. Cela pourrait expliquer la légère amélioration à mesure que le nombre de simulations maximum augmente.

5.5.2.2. Configuration défavorable

Pour ces expériences, nous avons progressivement réduit le nombre de simulations disponible pour le programme BHRF ; tout en conservant un nombre de simulations constant de 10000 pour le programme de référence. Comme nous pouvons l'observer dans la figure 5.10, le programme avec BHRF bat le programme de référence dès lors que celui-ci dispose de la moitié du nombre de simulations. Cependant pour ce même nombre de simulations, le programme avec BHRF nécessite 2 fois plus de temps que le programme de référence. Comme nous l'avons évoqué précédemment, l'acquisition de connaissances supplémentaires à la volée peut s'avérer exigeant en temps d'exécution. Des efforts supplémentaires aussi bien sur le modèle que sur l'implémentation doivent être consentis afin de rendre le programme compétitif à temps constant.

5.5.2.3. Limitation de la profondeur maximum

Par défaut, BHRF ne fixe aucune limite dans la taille maximale des schémas couverts. La taille des schémas couverts est bornée à tout instant par la profondeur atteinte par l'arbre de recherche. Dans ces expériences, nous avons choisi de limiter la taille maximale des schémas *a priori* afin de mesurer une telle influence.

PARAMÈTRES
△ Plateau= `9x9,19x19` ▼ Simulation= `10k` ■ Politique experte= `False`
□ ϵ= `100` ♦ Profondeur Max.= 2,5,∞ ◎ Volatile= `False`

Profondeur	**2**	**5**	∞
goban 9x9	$+14.8\% \pm 1.7$	$+17.5\% \pm 1.9$	$+17.3\% \pm 1.6$
goban 19x19	$+21.4\% \pm 4.2$	$+24.5\% \pm 3.4$	$+24.3\% \pm 3.5$

Table 5.2.– Victoires d'une politique de *roll-out* avec BHRF contre une purement aléatoire

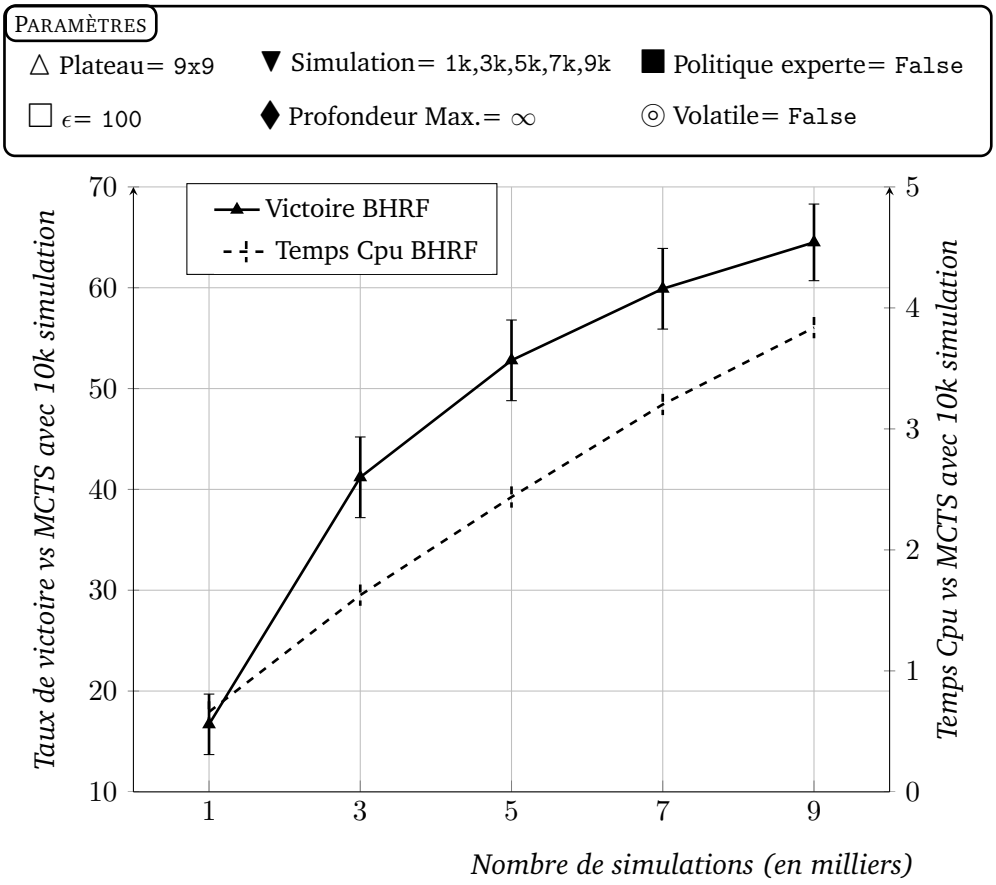


Figure 5.10.– Taux de victoire et temps d'exécution de BHRF avec un nombre variable de simulations contre le programme de référence avec un nombre fixe de 10000 simulations.

La table 5.2 montre que les performances du système augmentent pour des schémas temporels de plus en plus grands [12]. Au-delà d'une taille maximale de 5, BHRF ne semble plus apporter de bénéfice supplémentaire. En effet, pour des tailles trop grandes, les schémas temporels appris ont peu de chances d'apparaître lors des *roll-out*. Les travaux réalisés par Drake sur LGRF suggéraient que des contextes d'une taille maximale de 2 suffisaient. Pour BHRF, nous pensons que cette limitation gagne à être légèrement supérieure comme en témoignent les résultats pour une profondeur égale à 5. Toutefois l'utilisation de schémas plus grands ne nuit pas pour autant aux performances, à nombre de simulations constant. C'est pourquoi nous ne fixerons plus la profondeur maximale par la suite.

5.5.3 Qualité des connaissances auto-acquises

La plupart des politiques de *roll-out* construites à la volée sont mises à jour indépendamment de l'arbre de recherche. Une des différences majeures de BHRF consiste

12. Cf. note 10

à réutiliser les mises à jour de l'arbre pour influencer en retour le *roll-out*. Dans cette section, nous nous intéresserons aux aspects qualitatifs des connaissances ainsi acquises afin de mieux saisir la dynamique de BHRF : à quels types d'action correspondent les schémas appris ? quelles différences avec ceux présents dans une politique experte ?

Les valeurs contenues dans l'arbre de recherche bénéficient d'une politique de *roll-out* plus vraisemblable : c'est à dire disposant d'un minimum de connaissances élémentaires du jeu à l'instar d'un joueur humain débutant (cf. paragraphe *prior knowledge* dans la section 4.2.1.2). Les politiques *sequence-like* reproduisent un tel comportement dans des programmes jouant à un niveau professionnel comme *Fuego*. La plupart des attaques au jeu de Go implique une réponse locale (proche spatialement). Ainsi ces politiques considèrent tout d'abord les intersections autour du dernier coup joué afin de favoriser des réponses proches de celui-ci. Afin de sélectionner les réponses locales les plus pertinentes, ces politiques mobilisent dans un deuxième temps des connaissances tactiques comme les schémas spatiaux 3x3 (cf. Définition 3.11 p.38). Dans cette section nous évaluerons la qualité des actions produites lors des phases de *roll-out* à l'aune d'un tel comportement. Plus précisément, nous définirons un profil de *roll-out* d'après les deux critères suivants illustrés par la figure 5.11 :

— La capacité à jouer localement : une action est dite locale si elle est jouée dans une distance de Manhattan [13] d'une valeur de trois ou moins de l'action précédente ;

— La capacité à jouer localement des schémas spatiaux : une action correspond à un schéma spatial et local si l'action est locale et aussi bien l'action que les pierres autour coïncident avec un des schémas spatiaux utilisés dans les politiques *sequence-like* .

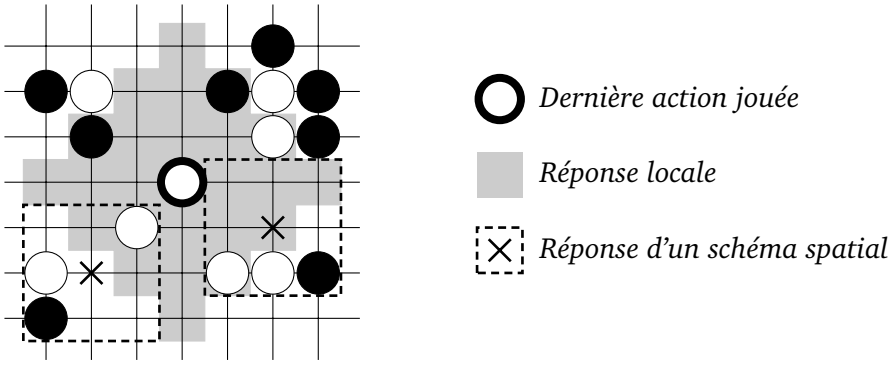


Figure 5.11.– Critères pour les profils de *roll-out*

13. La distance de Manhattan revient à calculer le plus court chemin séparant les deux positions des actions en suivant les lignes du quadrillage.

Afin d'obtenir le profil des actions produites par des schémas de BHRF exclusivement, nous avons désactivé la politique de *roll-out* experte et fixé l'utilisation de BHRF à son maximum ($\epsilon = 100$). Ainsi, chaque fois que la séquence des actions correspond au contexte de l'un des schémas de BHRF, la réponse associée est effectivement jouée conformément à sa valeur UCT. En pratique les schémas fournis par BHRF sont employés pour 51% des actions en moyenne. Nous comparons le profil des actions de BHRF avec deux autres profils :

— Un profil de *roll-out* aléatoire, obtenu avec le programme *Fuego* sans activé la politique de *roll-out* experte ;

— Un profil de *roll-out* expert, obtenu avec le programme *Fuego* dans sa version par défaut (avec la politique de *roll-out* experte).

Dans le programme *Fuego* en particulier, les réponses locales et l'emploi de schémas spatiaux 3x3 sont privilégiés aussi bien dans la politique de *descent* que dans la politique de *roll-out* : dans la politique de *descent* pour initialiser les valeurs des noeuds et dans la politique de *roll-out* pour générer les actions. À la différence des politiques de *roll-out* expertes, la politique avec BHRF ne repose pas sur des règles fournies préalablement au programme. Ces expériences sont l'occasion d'observer dans quelle mesure la politique avec BHRF reproduit une politique *sequence-like*.

5.5.3.1. Capacité à jouer localement

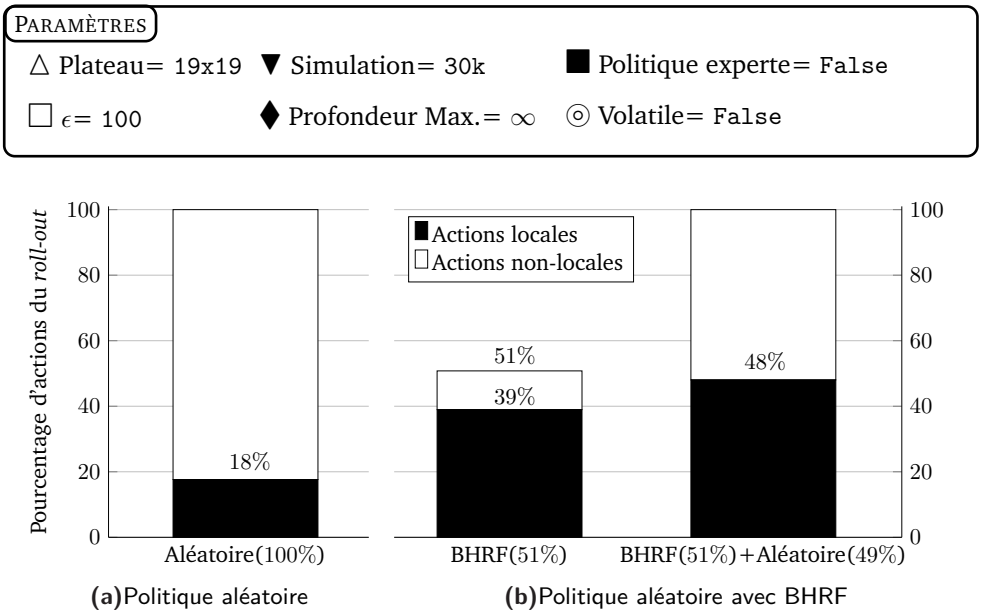


Figure 5.12.– Profil de *roll-out* : actions locales (politique aléatoire vs politique BHRF)

Tout d'abord, nous comparons le profil de BHRF avec celui d'une politique aléatoire dans la figure 5.12. Nous observons très clairement que la politique de *roll-out* avec BHRF joue bien plus localement qu'une politique purement aléatoire. Cette capacité

à jouer localement pourrait, selon nous, expliquer en grande partie les résultats contre un programme sans politique experte comme cela a été présenté dans la section 5.5.2.1.

D'après la figure 5.13, la politique avec BHRF joue en moyenne le même nombre d'actions locales qu'une politique experte (48%). Les politiques ne sont pas pour autant équivalentes. En effet la capacité de jouer localement dans une politique experte est définie explicitement *a priori* alors que BHRF la récupère à partir des informations présentes dans l'arbre. En outre toutes les actions locales ne sont pas nécessairement pertinentes. Dans les politiques *sequence-like*, les schémas spatiaux sont utilisés pour identifier les meilleures actions.

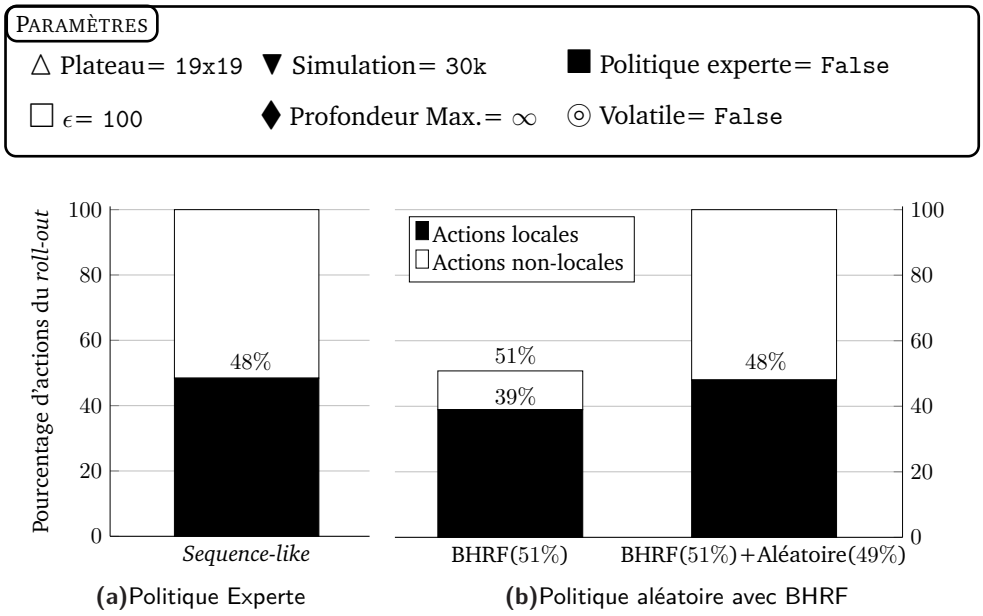


Figure 5.13.– Profil de *roll-out* : actions locales (politique experte vs politique BHRF)

5.5.3.2. Capacité à jouer localement des schémas spatiaux

La figure 5.14 indique que la politique avec BHRF produit jusqu'à 4 fois plus d'actions coïncidant avec un schéma spatial expert qu'une politique purement aléatoire. Ainsi BHRF a aussi réussi à extraire de l'arbre de recherche une partie des schémas spatiaux utilisés notamment pour initialiser les noeuds de l'arbre. Cependant à la différence de schémas experts directement introduits dans la politique de *roll-out*, ceux-ci ont été filtrés à la volée (les mieux évalués ont plus de chances d'être joués). Ainsi, moins d'un tiers seulement des actions provenant d'un schéma temporel de BHRF correspondent à un schéma spatial expert.

La politique *sequence-like* joue encore plus de schémas spatiaux experts comme nous pouvons le voir dans la figure 5.15. Cela n'est pas surprenant car l'utilisation de

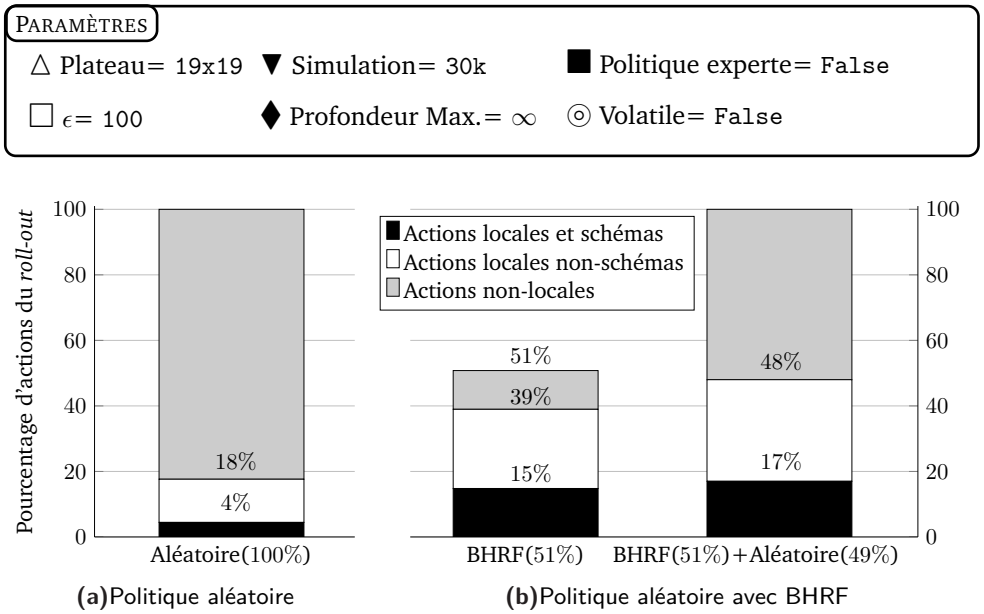


Figure 5.14.– Profil de *roll-out* : schémas spatiaux locaux (politique aléatoire vs politique BHRF)

ces schémas spatiaux est systématique chaque fois que la situation se présente dans le programme *Fuego*. Comme toute politique statique, le choix de l'action ne tient pas compte du contexte actuel de la partie en cours. Par l'intermédiaire des valeurs contenues dans l'arbre de recherche, BHRF est plus susceptible de reconsidérer l'emploi de ses schémas temporels, voire même en obtenir de nouveaux spécifiques à la situation courante.

5.5.4 Politique de *roll-out* experte avec BHRF

L'analyse qualitative des *roll-out* de la précédente section 5.5.3.2 a montré que les actions jouées par BHRF sont locales mais ne correspondent pas nécessairement aux schémas experts systématiquement. À défaut de les reproduire, les schémas temporels couverts par BHRF peuvent compléter les schémas experts des politiques expertes. C'est pourquoi dans cette section, nous avons choisi d'intégrer la politique experte à la politique actuelle avec BHRF. Lorsqu'aucune action proposée par BHRF n'a été appliquée, les règles expertes employées par la politique de *roll-out* de *Fuego* sont appliquées. Le programme utilisant BHRF affronte ici le programme *Fuego* par défaut (disposant d'une politique de *roll-out* experte) sur un goban 19x19. Le nombre de simulation maximum a été fixé à 30000 afin d'accumuler suffisamment d'expériences à propos de la partie courante. Dans une telle configuration, le programme avec BHRF s'exécute 12 fois plus lentement que le programme *Fuego*.

Nous pouvons voir dans la figure 5.16 que le programme avec BHRF l'emporte significativement sur le programme *Fuego* lorsque BHRF est employé modérément

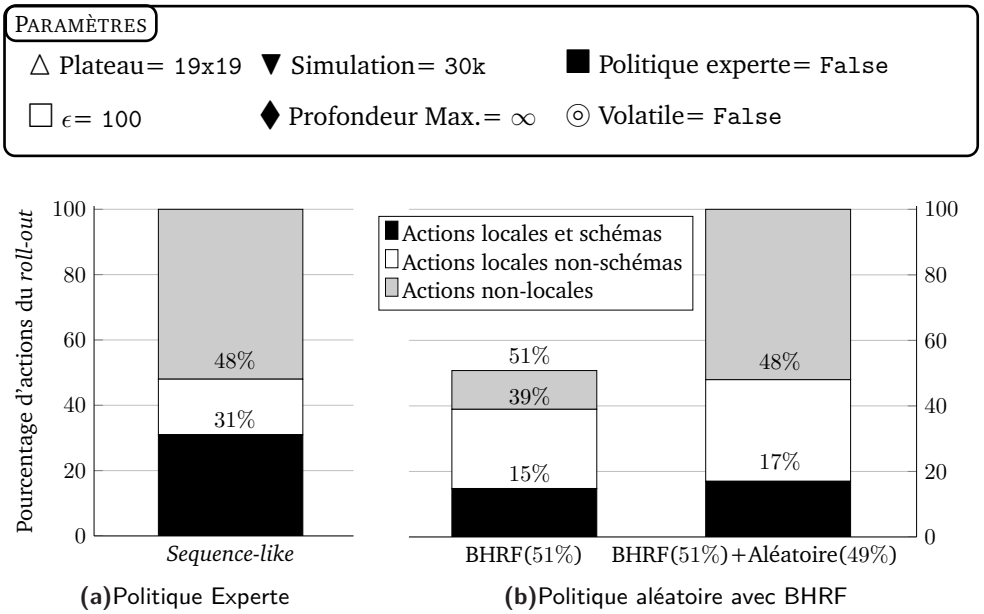


Figure 5.15.– Profil de *roll-out* : schémas spatiaux locaux (politique experte vs politique BHRF)

($\epsilon = 15$). Ainsi l'emploi de quelques schémas temporels bien choisis peuvent grandement améliorer les performances du programme. À l'inverse cela signifie aussi qu'un emploi trop systématique des schémas temporels affecte l'apprentissage global du système.

Les politiques de *roll-out* expertes utilisées par défaut dans *Fuego* jouent en priorité autour du dernier coup joué. Sur un goban 9x9, les combats locaux traversent rapidement la totalité du plateau mais sur un goban 19x19, les combats locaux doivent parfois tenir compte de régions du plateau bien plus éloignées. Comme nous l'avons mentionné dans la section 5.5.3, les schémas couverts par BHRF comprennent aussi bien les représentations expertes que celles auto-acquises par l'arbre de recherche. La dynamique de BHRF permet d'ajuster la localité des actions suivant les états couverts dans l'arbre de recherche. Ce faisant, la politique de *roll-out* bénéficie de connaissances *a priori* mais adaptées à la situation présente.

5.5.5 Étude de la version volatile de BHRF

BHRF améliore les performances du programme *Fuego* sur un goban 19x19. Ce gain à nombre constant de simulations est réalisé au détriment d'un temps d'exécution bien plus long. Un des enjeux de la version volatile de BHRF est d'accélérer la politique d'apprentissage en simplifiant la procédure de sélection (cf. section 5.4).

Dans cette section nous étudierons dans quelle mesure la simplification de la politique de *roll-out* atténue les performances de BHRF. Le programme sera testé dans des les mêmes conditions que la section précédente. Les expérimentations seront réalisées

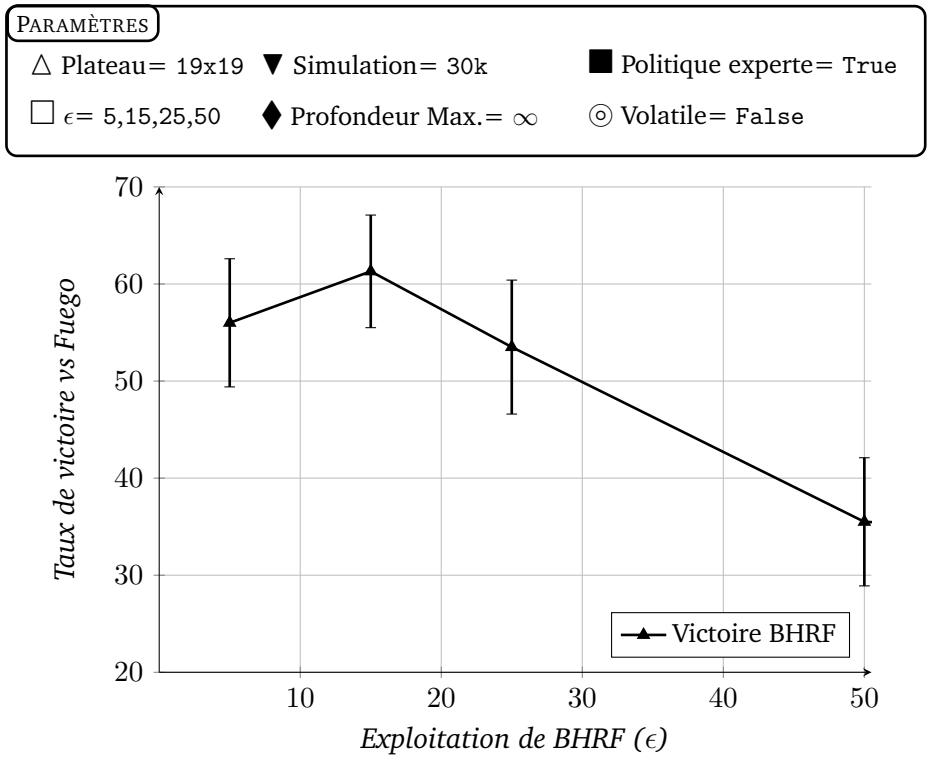


Figure 5.16.– Taux de victoire en 19x19 avec des politiques de *roll-out* à l'état de l'art

sur un goban 19x19 avec un nombre maximum d'épisode égal à 30000. Les politiques de *roll-out* expertes seront activées pour les deux programmes. Dans une telle configuration, le programme avec la version volatile de BHRF s'exécutera moins de 2 fois plus lentement que le programme *Fuego* [14]. La figure 5.17 regroupe les résultats obtenus pour différentes valeurs de ϵ et de profondeur maximale. Nous nous attacherons tout d'abord à comparer cette version volatile avec la version non-volatile étudiée précédemment. La version volatile de BHRF tire son inspiration du renforcement mis en oeuvre par la méthode LGRF[BD10]. Dans cette méthode le contexte maximal des schémas est fixé à 2. Nous verrons si une telle restriction se justifie pour BHRF aussi.

5.5.5.1. Comparaison entre les deux versions de BHRF

Lorsqu'aucune taille maximale de schéma n'est imposée, nous pouvons constater que la version volatile atteint des performances semblables à la version non-volatile : 61% de taux de victoire face au programme de référence *Fuego*. Dans la version volatile, ces performances sont atteintes pour une valeur plus grande de ϵ. Toutefois

14. La version volatile implémentée pour ces résultats utilisait un arbre direct de schémas mais les algorithmes n'étaient pas parfaitement optimisés. Nous pensons qu'il est encore possible d'accélérer plus encore un tel processus.

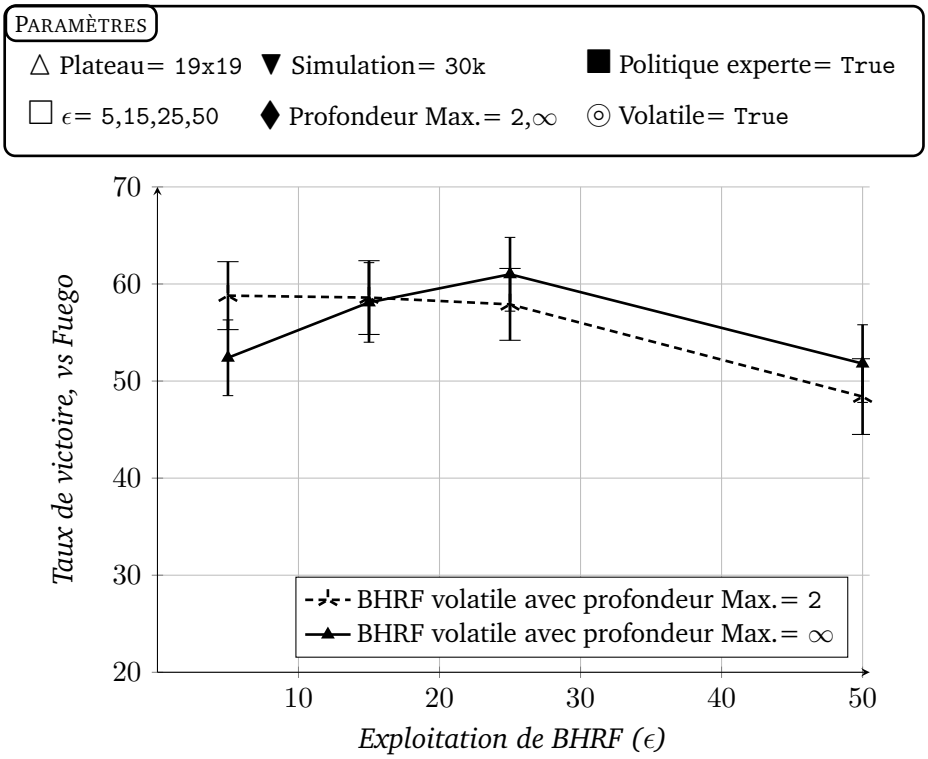


Figure 5.17.– Victoires de BHRF volatile contre *Fuego*

le paramètre ϵ ne reflète pas l'utilisation effective de BHRF dans chacune des deux versions.

La version volatile de BHRF détermine la réponse à jouer pour tout contexte en fonction des dernières actions sélectionnées par la politique de *descent*. De tels résultats suggèrent que le choix opéré par la politique de *descent* se substitue à une politique de sélection de réponse plus complexe. Cela ne signifie pas pour autant qu'une telle politique volatile sélectionne les meilleures réponses à chaque contexte. Comme nous l'évoquerons dans la section 5.6.2, nous pensons que d'autres dynamiques plus complexes permettent d'expliquer un tel résultat.

5.5.5.2. Influence de la profondeur maximale

Les résultats obtenus pour une profondeur maximale de 2 sont légèrement inférieurs à ceux obtenus sans limite imposée. Les meilleurs résultats pour une profondeur maximale de 2 sont obtenus pour une valeur bien plus faible de ϵ (une valeur de $\epsilon = 5$ contre une valeur de $\epsilon = 25$ sans aucune limitation). La politique de *roll-out* proposée pour la version volatile de BHRF permet d'expliquer en partie un tel résultat.

Contrairement à la version non-volatile de BHRF, les réponses proposées ne sont pas nécessairement légales. Le cas échéant, la politique experte est appliquée. Lorsqu'aucune limite n'est fixée, les schémas correspondant à des contextes plus grands sont systématiquement sélectionnés. Or de tels schémas ont moins de chance d'être mis à jour que des schémas plus petits. Ainsi, si une réponse non légale a été attribuée à l'un de ces schémas, elle ne sera changée que lorsque le même contexte apparaîtra à nouveau dans l'arbre de recherche. En fixant la taille maximale à 2, les plus grands schémas possibles restent maintenus à jour fréquemment et sont moins propices à fournir des réponses illégales.

5.6 Discussion

BHRF introduit une dynamique d'apprentissage complémentaire à MCTS. À l'instar de nombreuses méthodes proposées dans la littérature, cette dynamique porte sur l'élaboration d'une politique de *roll-out* adaptative. Cette politique dynamique repose néanmoins sur une structure de données dynamique : c'est à dire que de nouvelles représentations non fixées *a priori* vont être progressivement évaluées. Dans la section 5.6.1 nous comparerons BHRF avec d'autres méthodes de littérature proposant une structure de données dynamique pour le *roll-out*.

La particularité de BHRF par rapport aux méthodes évoquées tient à son renforcement. En effet, les représentations dynamiques de BHRF sont alimentées par les représentations dynamiques de l'arbre de recherche de MCTS. Ce faisant BHRF dépend de la dynamique de MCTS d'une part mais bénéficie aussi de l'ensemble des heuristiques développées pour la politique de *descent*. De telles interactions sont complexes à analyser. Dans la section 5.6.2, nous nous intéresserons en particulier aux interactions entre BHRF et MCTS. Nous discuterons ensuite des interactions entre BHRF et des connaissances de long terme dans la section 5.6.3.

5.6.1 Arbre de recherche local pour la politique de *roll-out*

La conception de politique de *roll-out* adaptative sur le modèle de MCTS a fait l'objet de nombreuses recherches. L'objectif à terme était généralement de concevoir un arbre de recherche local pour la phase de *roll-out* [Bau11 ; Bai10]. BHRF s'inscrit pleinement dans cette perspective. La politique présentée s'adapte suivant l'évolution de la partie et l'arbre direct de schémas, présenté dans la section 5.3, s'approche de l'idée d'un arbre local de recherche. Dans cette section, nous présenterons deux méthodes proches de BHRF sur ces aspects ci : le *Move answer tree* de Baier [Bai10] et le *loose-tree* spatial proposé par Basalduà et al. [Bas+14]. Ces deux méthodes se distinguent notamment de BHRF par leur renforcement construit à partir des actions

sélectionnées dans le *roll-out*. Nous reviendrons plus longuement sur cet aspect lors de la prochaine section. Pour finir, nous émettrons une hypothèse quant à l'intérêt de développer des politiques de *roll-out* adaptative pour MCTS.

5.6.1.1. *Move answer tree*

Le *Move answer tree* a été proposé par Baier dans son mémoire de Master. Cette structure de données est très proche de l'arbre direct de schémas présenté en Section 5.3.2. Chaque noeud du *Move answer tree* porte un schéma temporel et le noeud racine correspond au contexte vide. À la différence de l'arbre direct de schéma, il n'existe pas dans *Move answer tree* de relation entre les branches au travers des noeuds de retour. En effet l'extension du *Move answer tree* n'ajoute pas pour une même réponse, les schémas pour contextes de tailles différentes comme dans BHRF. Ce faisant, cette structure de données ne vérifie pas la propriété 5.2.

Le *Move answer tree* a été développé pour le jeu de Go sur le programme Orego. Les résultats obtenus n'étaient pas significatifs face au programme Gnugo. Outre le renforcement différent (le *Move answer tree* est renforcé à partir des actions du *roll-out*), nous pensons que les mauvais résultats obtenus dépendent aussi de la politique d'apprentissage. En effet l'utilisation des schémas temporels n'était pas contrôlée par un paramètre comme ϵ. Les schémas temporels appris sont joués systématiquement lorsqu'il apparaissent. Or les meilleurs résultats obtenus avec BHRF ont été obtenus pour des utilisations mesurées des schémas temporels ($\epsilon = 15$ voire ϵ pour la version volatile).

5.6.1.2. *Loose-tree* spatial

Dans l'article de Basalduà et al. [Bas+14], leur programme évalue de nombreux schémas spatiaux à la volée. Afin d'organiser ces schémas entre eux, ils proposent une structure qui s'apparente à un arbre de recherche sans pour autant être explicitement réifié : un arbre « lâche » (*loose tree* en anglais) [15]. Un noeud correspond à un schéma spatial couvert par leur méthode. Des arêtes valuées relient deux noeuds deux à deux lorsque deux schémas spatiaux (contexte et réponse) se sont succédés.

Cette structure de données est essentiellement spatiale, à la différence de la forêt rétrograde de schémas ou du *Move answer tree*. Comme nous l'avons évoqué dans la section 3.1.2.2, les schémas spatiaux se prêtent mieux au jeu de Go que les schémas temporels. En revanche, il est difficile de distinguer dans les schémas spatiaux si

15. L'article ne propose pas de nom pour identifier la structure de données en question. Seul le terme de *loose-tree* est employé en guise d'analogie. Nous conserverons ce terme pour identifier la structure.

l'évaluation se rapporte au contexte ou à la réponse comme l'a montré Lew dans son mémoire de doctorat[Lew11] [16]. Il semblerait néanmoins que ce phénomène ne soit pas un frein pour le moment à cette approche ; en témoigne les gains de performances qu'il a obtenu pour son programme de Go de référence [Bas+14] (environ $+15\%$ de victoire en plus).

5.6.2 Interaction entre BHRF et MCTS

La spécificité de BHRF par rapport aux précédentes approches est de réutiliser des connaissances provenant de l'arbre de recherche. Les seules approches similaires dans la littérature mobilisent des représentations bien plus élémentaires généralement limitées à la simple action (cf. section 4.2.2.2). Une telle approche permet notamment de renforcer les liens entre les phases de *descent* et *roll-out*. Ce type d'interaction combiné avec l'emploi de représentations floues induit des dynamiques complexes dont l'étude reste encore à approfondir. Nous évoquerons tout d'abord l'une de ces interactions qui mériterait une plus longue étude. Ensuite, nous avancerons l'idée de fusionner les deux phases au travers d'une structure de données commune et dans quelle mesure BHRF réalise une telle perspective.

5.6.2.1. Interactions complexes entre les deux politiques

Les phases de *descent* et de *roll-out* sont différentes sous bien des égards et les politiques associées le sont tout autant. Néanmoins la politique de *roll-out* entretient des liens plus ou moins directs avec la politique de *descent*. En effet la politique de *roll-out* influence le cours de l'épisode et participe indirectement au renforcement de la politique de *descent*. Cette influence devient encore plus prégnante lorsque le renforcement des évaluations dépend des actions sélectionnées lors du *roll-out* comme c'est la cas pour les mises à jour de l'amélioration RAVE (cf. section 4.2.1.4).

À l'inverse, BHRF établit un lien direct entre la politique de *descent* et la politique de *roll-out*. Ce faisant, les interactions entre les deux politiques deviennent alors plus complexes. D'une part BHRF agit en tant que catalyseur de l'apprentissage en accélérant les renforcements des évaluations RAVE. D'autre part la politique de *descent* induite par RAVE alimente BHRF. Un tel phénomène pourrait expliquer le succès de BHRF et il serait intéressant de le quantifier.

Nous avons fait des expérimentations préliminaires allant dans ce sens. En désactivant RAVE pour les deux programmes, nous limitons l'influence entre la politique de *roll-out* et celle de *descent*. Les performances de la version volatile de BHRF en

16. cf. note 2 p.101

particulier chutent de $58.1 \pm 4.2\%$ à $46 \pm 6\%$ pour une configuration semblable à celle employée dans la section 5.4 (avec $\epsilon = 15$). Toutefois il reste à identifier si cette baisse est bien due à l'absence de ces renforcements supplémentaires ou bien si la nouvelle politique de *descent* sans RAVE alimente BHRF avec de mauvais schémas temporels. Des expérimentations complémentaires doivent encore être réalisées dans ce sens.

5.6.2.2. Vers une politique commune aux deux phases

Les résultats obtenus par BHRF suggèrent que les deux politiques de MCTS gagnent à interagir ensemble. Une perspective de recherche possible serait de fusionner à terme les deux politiques en une seule. Nous pensons qu'une telle évolution de MCTS lui permettrait de pallier les limitations intrinsèques à sa représentation comme l'effet d'horizon (cf. Définition 3.8 p.36).

Dans un premier temps, il sera nécessaire de proposer une structure de données commune aux deux phases. L'arbre de recherche est une structure de données tellement précise qu'il n'est pas envisageable de l'employer tout au long d'un épisode. L'arbre direct de schémas au contraire permet de couvrir des représentations floues pour la phase de *roll-out* mais aussi des représentations précises pour la phase de *descent*. En effet un schéma temporel devient une représentation précise lorsque le contexte temporel remonte jusqu'au début de la partie. Ainsi l'arbre direct de schéma réalise la transition d'une représentation précise à une représentation plus floue (couvrant ainsi les deux phases). La phase de *roll-out* débuterait dès qu'il sera nécessaire d'emprunter le premier noeud de retour.

Toutefois l'extension du renforcement de BHRF à l'ensemble de l'épisode poserait d'évidents problèmes en terme de mémoire. Conformément à la mise à jour proposée, les actions produites au cours des deux phases *descent* et *roll-out* donneraient lieu à un nouveau schéma. En effet les représentations de BHRF sont trop brutes et ne se concentrent pas sur les aspects les plus importants à mémoriser : elles ne sont pas encore suffisamment synthétiques. Pour ce faire, il serait nécessaire d'abstraire la représentation acquise comme nous le verrons dans la section suivante.

5.6.3 Portée des connaissances de BHRF

Les connaissances apprises par MCTS sont très contextuelles. Ces connaissances sont apprises à la volée pour la situation courante et seront rapidement obsolètes dans les tours à venir. Ces connaissances ne seront pas pérennes. Des connaissances valables sur plusieurs tours, voire sur plusieurs parties, nécessitent un apprentissage bien plus long. Cet apprentissage peut être réalisé par apprentissage supervisé (cf.

section 3.2.2) ou bien par conception experte lorsque l'expert humain arrive à formaliser ses propres connaissances pour un langage machine (cf. section 3.2.1).

BHRF propose d'utiliser les expériences simulées afin de capitaliser des connaissances sur plusieurs tours. Cependant à la fin de la partie, l'ensemble de ces connaissances sont oubliées. Dans cette section, nous nous intéresserons à la capacité de BHRF à produire des connaissances plus pérennes et interagir avec elles. Nous montrerons tout d'abord pourquoi des représentations comme celles utilisées par BHRF ne se prêtent pas à des connaissances valables pour plusieurs parties. En revanche, comme nous le verrons dans la deuxième section, BHRF a la capacité de s'approprier au cours de l'apprentissage de connaissances plus pérennes à l'instar les schémas experts.

5.6.3.1. Élargir les représentations accumulées par BHRF

Les représentations apprises par BHRF interviennent lors de la phase de *roll-out*. Les connaissances mobilisées pour cette phase doivent satisfaire un double impératif. Il est nécessaire d'une part que ces connaissances soient pertinentes pour le contexte dans lequel elles sont appliquées et d'autre part quelles soient exploitables dans un grand nombre de situations.

Cependant, les contraintes imposées par les schémas temporels sont très restrictives. Pour une taille donnée, l'ensemble des actions du contexte doivent avoir été jouées au préalable et de façon consécutive. Il suffit qu'une action diffère ou bien que l'ordre de ces actions soit changé pour que le schéma ne soit pas valide. Dans le cadre du jeu de Go, de nombreux schémas deviennent obsolètes dès lors qu'une action présente dans celui-ci a été jouée.

Afin d'élargir l'emploi des schémas temporels appris, il serait envisageable de lever partiellement certaines contraintes imposées par ceux-ci, au risque de rendre leur mise à jour ou leur application plus coûteuse en temps de calcul. Par exemple, il serait possible d'autoriser dans le contexte plusieurs actions différentes pour la même position, à l'instar des schémas spatiaux ou bien d'autoriser que les actions présentes dans le contexte ne soient pas strictement consécutives.

Une autre perspective plus ambitieuse serait d'abstraire les représentations ; en d'autres termes, de s'éloigner de représentations brutes au profit de représentations plus enrichies. Les représentations plus enrichies s'appliquent à un bien plus grand nombre de situations. Le choix des abstractions introduites a d'importantes conséquences sur la pertinence des représentations. À défaut d'identifier les abstractions pertinentes, il est possible de les apprendre. Un tel processus n'est réalisable que par l'intermédiaire de structures de données suffisamment souples et expressives dans le

même temps. À notre connaissance, seuls les réseaux de neurones offrent de telles capacités mais leur fonctionnement précis reste encore difficile à comprendre (cf. Définition 3.14 p.41)

5.6.3.2. Interaction entre BHRF et des connaissances pérennes

Les programmes de niveau professionnel comme *Fuego* mobilisent des connaissances expertes lors des deux phases : dans la politique de *descent* pour initialiser des évaluations et dans la politique de *roll-out* telles quelles, pour générer les actions. Dans les deux cas, ces connaissances sont définies *a priori* à partir d'un nombre restreint de règles formelles. Contrairement à politique de *roll-out* qui est statique, la politique de *descent* peut contrebalancer le biais introduit à partir des résultats des simulations en d'autres termes, malgré cet *a priori*, la politique de *descent* peut finalement décider de ne plus privilégier des connaissances expertes si leur valeur de l'état associé est faible.

Les renforcements de BHRF sont les mêmes que ceux de la politique de *descent*. Dans la section 5.5.3, nous avons montré que la version non-volatile de BHRF retrouve bien les connaissances fournies *a priori* à l'arbre de recherche. Cependant ces connaissances auront été filtrées par les passes successives de la politique de *descent* et de ses renforcements. Ce faisant, la pertinence des connaissances expertes est adaptée à la situation courante. Ainsi BHRF s'approprie de connaissances plus pérennes au travers des renforcements induits par la politique de *descent*. En d'autres termes, le système réussit à accommoder, par l'intermédiaire de BHRF, les connaissances expertes dont il dispose *a priori* sur le problème.

5.7 Synthèse

Au cours de chapitre nous avons présenté une dynamique d'apprentissage complémentaire de MCTS. Nous avons montré qu'une meilleure assimilation des connaissances apprises par MCTS peut améliorer les performances globales du système. Comme nous l'avons évoqué au cours des chapitres précédents, les représentations accumulées dans l'arbre de recherche ne sont pas sujettes à être ré-exploitées. Nous pensons qu'une perspective prometteuse consiste à considérer MCTS comme un agent apprenant. En effet, une meilleure assimilation des connaissances acquises permet d'éviter d'apprendre à nouveau des schémas déjà présents dans différentes branches de l'arbre par exemple. L'analyse poursuivie tout au long de ce chapitre considère MCTS et BHRF comme faisant partie d'un même processus d'apprentissage par renforcement. Nous pensons qu'une telle démarche donne un meilleur aperçu

de comment le système considère son expérience simulée et offre de nouvelles perspectives de recherche pour MCTS.

La représentation dynamique présentée dans ce chapitre permet de mémoriser des connaissances à la volée issues de l'arbre de recherche. Les résultats obtenus indiquent que BHRF capture bien les connaissances qui y sont présentes ; notamment celles fournies *a priori*. De plus les schémas appris complètent les connaissances expertes habituellement utilisées pour le *roll-out*. Ainsi, le programme *Fuego* complété par BHRF améliore ses performances de 11%. Ces résultats montrent le potentiel d'une telle approche bien que le fort ralentissement induit par le processus d'apprentissage complémentaire empêche toute expérimentation à temps constant. Néanmoins des efforts ont été réalisés ensuite pour améliorer le temps d'exécution. Tout d'abord nous avons proposé une structure de données optimisée pour la mise en application des connaissances accumulées. Ensuite nous avons proposé une seconde amélioration pour simplifier le processus de décision de BHRF. Les choix réalisés par la politique de *descent* sont récupérés au travers d'évaluations volatiles afin d'accélérer la politique de BHRF. Avec cette approche, des résultats similaires ont été obtenus tout en demandant bien moins de temps de calcul. Les efforts doivent encore être poursuivis afin de mieux cerner les raisons d'un tel succès.

Nous avons choisi d'appliquer cet algorithme au jeu de Go car il s'agit d'un problème nécessiteux en terme de connaissances. Néanmoins le modèle actuel est applicable à n'importe quel autre jeu. Une implémentation plus efficace pourrait tenir compte de caractéristiques spécifiques au jeu considéré mais dépasserait le cadre du travail réalisé ici.

6 Réification d'objectifs intermédiaires dans la dynamique d'apprentissage

La méthode MCTS poursuit tout au long de la partie le même objectif final : maximiser ses chances de victoire. La poursuite de cet objectif s'inscrit dans une dynamique d'apprentissage spécifique par le choix notamment des récompenses perçues en fin de simulations. Si les évaluations apprises dans l'arbre évoluent au fur et à mesure des simulations, la dynamique d'apprentissage reste la même quelle que soit la situation rencontrée. Au cours de ce chapitre, nous envisageons la possibilité pour MCTS de changer momentanément de dynamique d'apprentissage au cours de la recherche. Dans le modèle proposé, chaque dynamique d'apprentissage répond à un objectif différent avec sa stratégie associée et sa rétroaction spécifique. En particulier, nous considérons ici la réification d'objectifs intermédiaires permettant ainsi à l'agent de structurer sa recherche sur un plus long terme. En outre, l'agent apprend dans le même temps à évaluer chacune de ces stratégies au travers d'une dynamique d'apprentissage complémentaire et adapte ainsi l'objectif poursuivi suivant la situation.

Cette extension de la dynamique d'apprentissage constitue la deuxième contribution de cette thèse. Dans la section 6.1 nous motivons cette contribution à partir d'exemples tirés de la littérature. Ensuite nous présentons dans la section 6.2, le modèle général de MCTS avec des stratégies locales spécifiques. Ce modèle est ensuite instancié pour les jeux de Go et du Clobber Solitaire Impartial dans la section 6.3. Nous présentons systématiquement les objectifs intermédiaires retenus, ainsi que les résultats obtenus. Nous montrons notamment que le modèle proposé réussit avec succès à orienter la recherche autour d'objectifs intermédiaires.

6.1 Motivations

La méthode MCTS oriente l'exploration de l'espace de recherche au travers d'une dynamique d'apprentissage. L'interaction entre les valeurs présentes dans l'arbre de recherche et la politique associée $\pi_{descent}$ constitue le « moteur » de la méthode. Cette dynamique est définie *a priori* et n'est pas en mesure de s'adapter à des situations nécessitant des dynamiques spécifiques comme nous le présentons dans la

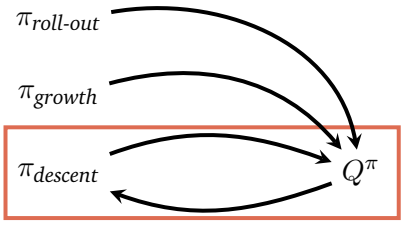


Figure 6.1.– Dynamique d'apprentissage de MCTS

section 6.1.1. Dans le domaine de l'IA, la réalisation de tâches spécifiques au cours d'un processus complexe revient à réifier des objectifs intermédiaires. Nous revenons sur cette notion dans la section 6.1.2 en précisant comment elle est traitée dans les programmes de Go mais aussi en apprentissage par renforcement.

Une perspective pour adapter fortement l'orientation de dynamique d'apprentissage de MCTS consiste à intégrer la poursuite d'objectifs intermédiaires au cours de l'apprentissage. Nous évoquons cette perspective dans la section 6.1.3, que nous allons ensuite préciser progressivement au cours de ce chapitre.

6.1.1 Limitations de la dynamique d'apprentissage de MCTS

Au cours de cette section, nous revenons sur la dynamique d'apprentissage de MCTS. Nous mettons tout d'abord en avant que la méthode ne dispose pas, en l'état, de mécanisme pour adapter sa dynamique d'apprentissage . Nous présentons ensuite un exemple de la littérature illustrant dans quelle mesure cette limitation est problématique et l'intérêt de s'en affranchir.

6.1.1.1. Dynamique d'apprentissage unique

La dynamique d'apprentissage restitue les interactions entre les processus de résolution et les représentations apprises. À propos de MCTS, nous avions identifié dans le chapitre 4 qu'elle comportait 3 politiques. Ces trois politiques sont toutes en interaction avec les mêmes représentations correspondant aux valeurs accumulées dans l'arbre de recherche. La politique de *descent* est, par défaut, la seule politique interdépendante avec les valeurs présentes dans l'arbre, comme cela est souligné en rouge sur la figure 6.1.

Comme nous l'avons évoqué dans la section 4.3.1.2, cette dynamique entre la politique $\pi_{descent}$ et les valeurs renforcées Q^{π} sera appliquée uniformément tout au long de la partie. Or il existe des situations pour lesquelles il convient de momentanément concentrer la recherche suivant des considérations plus spécifiques à la situation

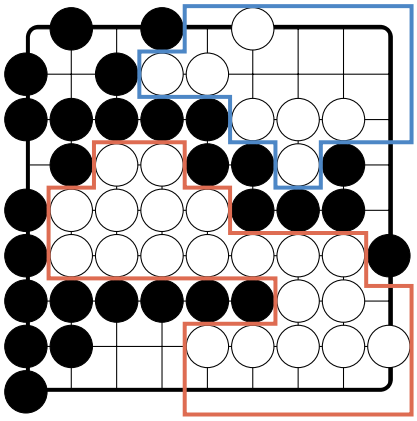


Figure 6.2.– Situation de Go difficile pour MCTS[HM13]

courante, comme nous le présentons ci-dessous. S'il existe des améliorations de MCTS pour concentrer la recherche à l'image des *GroupNode* (cf. section 4.2.1.1) ou du *prior knowledge* (cf. section 4.2.1.2), les critères pour approfondir ces exemples sont définis *a priori* et ne sont pas susceptibles d'évoluer au cours de la partie : le choix des critères ne participe pas à la dynamique d'apprentissage.

En d'autres termes, la dynamique d'apprentissage de MCTS dispose d'un mécanisme pour remettre en question les actions qu'il a choisies, au travers de la boucle entre $\pi_{descent}$ et Q^{π}, mais n'est pas en mesure d'ajuster fortement l'orientation de la dynamique qu'il exploite.

6.1.1.2. Limitations d'une recherche univoque

Müller et Huang[HM13] ont mis en avant certaines configurations de Go pour lesquelles les méthodes MCTS ne réussissent pas en appréhender les subtilités. En effet, ils ont constaté que les programmes de Go n'évaluent pas correctement les situations comportant deux sous-problèmes en parallèle. Pour ces configurations, le programme est généralement beaucoup plus optimiste sur sa situation, à cause des limitations inhérentes à la politique de *descent*.

La figure 6.2 est une configuration extraite de leurs exemples de tests proposés dans leur travail[HM13]. Dans cette situation, le programme de Go joue en tant que Noir et considère bien trop souvent, au cours de sa recherche, qu'il sera en mesure de capturer l'un deux groupes de pierres du joueur blanc et ainsi remporter la victoire. Cependant il se trouve que Blanc est en mesure de conserver ses deux groupes quoi que le joueur Noir fasse, s'il joue habilement. Cette situation est par conséquent irrémédiablement perdue pour le joueur Noir. Bien qu'il n'existe, en l'occurrence, plus aucun recours pour le joueur Noir, cette situation permet d'illustrer les limitations à poursuivre une recherche exclusivement globale.

D'un point de vue combinatoire, l'effet d'horizon appliqué à l'état du groupe (vivant/-mort) trompe l'estimation MCTS pour cette situation. À approfondir la recherche uniformément, la politique de *descent* passera indifféremment de l'un à l'autre, entraînant systématiquement l'exploration des différentes actions possibles. La phase de *roll-out* débutera sans avoir identifié le problème. Une situation aussi complexe sera livrée au hasard de la politique de $\pi_{roll\text{-}out}$, aboutissant généralement à la capture inopinée par Noir de l'un des deux groupes blancs.

Un joueur humain, bien que doté de moindres capacités de calcul, pourra cependant rapidement estimer de façon fiable l'état des groupes. Le joueur humain est ainsi plus efficace de part ses connaissances expertes accumulées, mais également grâce à sa capacité à recentrer la recherche localement et à la dédier à l'étude de l'état d'un groupe. En agissant ainsi, l'effet d'horizon se « localise » au bénéfice d'une meilleure estimation locale.

Sans pour autant prétendre lever intégralement les verrous identifiés pour la méthode MCTS, notre proposition a pour objectif d'ajuster la dynamique de la politique de *descent* grâce à la réification d'objectifs intermédiaires.

6.1.2 Réification d'objectifs intermédiaires dans la littérature

D'un point de vue cognitif, la réalisation de tâches spécifiques pour un problème plus vaste revient à réifier des objectifs intermédiaires, qui n'étaient pas formalisés jusqu'alors. La réification d'objectifs intermédiaires représente un thème classique dans l'intelligence artificielle, pour traiter de problèmes complexes. Dans cette section nous montrons brièvement comment ce thème a été traité dans les programmes de Go mais aussi, dans un second temps, comment il a été introduit pour le domaine de l'apprentissage par renforcement.

6.1.2.1. Objectif final et intermédiaires dans les programmes de jeux

L'objectif final (ou global) est celui poursuivi tout au long de la partie par n'importe quel joueur : remporter la victoire. Les objectifs intermédiaires (ou locaux) proposent une décomposition de la résolution d'une tâche complexe en parties plus facilement appréhendables. Ces objectifs sont généralement identifiés *a priori* et orientent la recherche du joueur.

Dans MCTS, les évaluations accumulées dans l'arbre Q^{π} ainsi que la politique de *descent* poursuivent exclusivement l'objectif global. Pour chaque noeud de l'arbre, toutes les actions possibles sont comparées au regard de ce même objectif. À l'inverse, les joueurs humains de Go ne considèrent jamais uniformément toutes les actions

possibles, ils restreignent immédiatement leur attention sur certaines zones selon des considérations stratégiques. Avant l'avènement de MCTS, les programmes de Go ne réalisaient jamais de recherche globale [BC01]. Suivant la configuration du jeu, plusieurs recherches arborescentes étaient réalisées autour d'objectifs intermédiaires, comme par exemple capturer un groupe ou connecter des chaînes de pierres. Les premiers programmes qui ont intégré des méthodes de Monte Carlo proposaient d'utiliser des simulations aléatoires pour évaluer des objectifs [CH05] mais, à notre connaissance, aucun programme n'a encore proposé d'intégrer des objectifs intermédiaires à MCTS.

6.1.2.2. Objectifs intermédiaires dans l'apprentissage par renforcement

La poursuite d'objectifs intermédiaires revient à changer momentanément la politique de recherche de l'agent jusqu'à ce que l'objectif poursuivi soit atteint. Le cadre formel de l'apprentissage par renforcement peut être étendu de manière à considérer des objectifs intermédiaires au cours de l'apprentissage. Les Processus Décisionnels Semi-Markovien (PDSM) généralisent les actions habituellement utilisées dans les PDM, par des options. Les options représentent indifféremment le choix d'une action ou bien l'application momentanée d'une autre politique de recherche. À l'image de n'importe quelle action classique, une option est évaluée et sélectionnée par une politique [1].

Définition 6.1: Option

Une option est la généralisation d'une action dans un PDSM. Une option o est définie par le triplet $< \mathcal{I}, \pi, \beta >$:

- $\mathcal{I} \subseteq \mathcal{S}$ est l'état à partir duquel l'option peut être initiée ;
- $\pi_o : \Omega \times \mathcal{A} \rightarrow [0:1]$, la politique dédiée de l'option ;
- $\beta : \Omega \rightarrow [0:1]$, la condition de terminaison de l'option.

Avec Ω l'ensemble des états et actions qui ont été sélectionnés depuis le début de l'option.

À partir du moment où une option est choisie, les actions suivantes seront décidées suivant la politique correspondante π_o (cf. Définition 3.15 p. 50), jusqu'à ce que la condition de terminaison probabiliste β soit vérifiée. Cette définition englobe notamment les actions habituellement employées dans les PDM. Il s'agit d'options élémentaires s'arrêtant systématiquement à l'étape suivante (la condition de terminaison est toujours égale 1).

1. Nous distinguerons néanmoins les actions des options introduisant une nouvelle politique, pour plus de clarté.

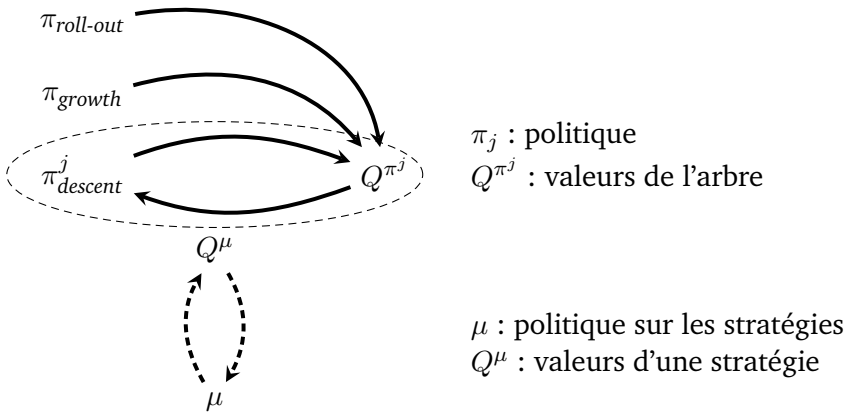


Figure 6.3.– Dyanmique d'apprentissage avec stratégie

Sutton, Precup et Singh ont montré que les options s'intègrent naturellement aux méthodes classiques d'apprentissage par renforcement [SPS99]. Elles s'appliquent donc naturellement à MCTS. Les options fournissent un cadre formel pour exprimer des objectifs intermédiaires dans un apprentissage par renforcement. Elles ne précisent ni la nature de l'objectif ni sa mise en oeuvre précise. Néanmoins les travaux de Sutton, Precup et Singh suggèrent des améliorations génériques à n'importe quelle mise en application de ces options.

6.1.3 Orienter la recherche autour d'objectifs intermédiaires dans MCTS

Dans MCTS, la dynamique d'apprentissage entre la politique $\pi_{descent}$ et les valeurs apprises Q^π poursuit un seul et même objectif global. Nous proposons ici de doter la méthode d'un panel de dynamiques d'apprentissage autour d'objectifs intermédiaires. Ces différentes dynamiques constituent autant de stratégies d'exploration locales, pour approfondir momentanément la recherche : chacune de ces stratégies poursuivra un objectif intermédiaire différent et la recherche sera menée dans un sous-arbre de l'arbre de recherche. Une stratégie réifie un objectif intermédiaire dans le cadre de MCTS.

Conformément aux PDSM, chaque stratégie correspond à une option proposant une politique de *descent* dédiée $\pi^j_{descent}$, depuis des noeuds de l'arbre. Dans le même temps, la politique de recherche serait guidée au travers d'évaluations spécifiques à l'objectif local poursuivi Q^{π^j}. Chaque stratégie dispose ainsi de sa propre boucle d'apprentissage auto-catalytique comme cela est résumé par la figure 6.3. Enfin chacune de ces stratégies est évaluée dans son ensemble à l'image de n'importe quelle autre action. Ainsi, une politique générale π_μ permettrait d'adapter la dynamique d'apprentissage à adopter suivant la situation. L'interaction entre la politique π_μ et les valeurs

de ces stratégies Q^{μ} se renforcent mutuellement au travers d'une autre dynamique d'apprentissage ; en d'autres termes, une méta-dynamique d'apprentissage.

Par exemple, dans une situation de Go comprenant trois groupes de pierres blanches, le joueur Noir pourra évaluer l'impact d'une attaque sur chacun des groupes en quelques coups, pour ensuite approfondir la stratégie locale qu'il aura jugée la plus utile. L'agent va pouvoir concentrer sa recherche autour d'un nombre réduit d'actions grâce aux évaluations localisées spécifiques, et gagner un temps de calcul précieux. Nous voyons déjà quelques questions à préciser dans le contexte de MCTS : quelles rétroactions pour une stratégie locale ? Sur quelle profondeur poursuivre cette stratégie ?

Nous discutons dans la suite de ce chapitre de différentes alternatives pour mettre en oeuvre la réification d'objectifs intermédiaires.

6.2 MCTS avec des stratégies locales

Comme nous l'avons évoqué au cours du chapitre précédent, nous souhaitons doter MCTS de la capacité d'adapter sa dynamique d'apprentissage suivant la situation. Dans ce modèle, nous proposons de doter MCTS d'un panel de stratégies différentes qu'il sera en mesure de changer à la volée suivant la situation. Ces stratégies seront autant de dynamiques d'apprentissage différentes poursuivant chacune un objectif différent. Outre la stratégie par défaut de MCTS poursuivant l'objectif final (c'est à dire gagner globalement la partie), les autres stratégies traiteront d'objectifs intermédiaires servant à la résolution de l'objectif final. Pour ce modèle, les stratégies sont toutes définies *a priori* mais nous espérons, à terme, que l'agent sera capable de les concevoir de lui-même ou de façon semi-automatique. En attendant, les stratégies locales devront être systématiquement précisées pour le problème traité.

L'intégration de stratégies locales à MCTS requiert un réaménagement de la dynamique d'apprentissage par défaut en une méta-dynamique d'apprentissage. Dans la section 6.2.1, nous présentons les principales modifications à apporter aussi bien au niveau des représentations qu'au niveau du déroulement d'un épisode. Nous revenons ensuite plus en détail, dans la section 6.2.2, sur la réification des objectifs intermédiaires dans les stratégies locales car cet aspect est déterminant pour le succès du modèle en pratique. Nous donnons un aperçu des principaux verrous ainsi que différentes possibilités pour les traiter. Enfin, nous évoquons diverses extensions possibles du modèle, proposé dans la section 6.2.3.

6.2.1 Méta-dynamique d'apprentissage pour MCTS

Le modèle proposé s'intègre assez naturellement à MCTS. À chaque objectif intermédiaire est associée une stratégie correspondante. Au début de chaque épisode, la méthode décide d'une stratégie à appliquer et approfondit la recherche suivant cette stratégie comme cela est résumé dans la figure 6.4. Dans les sections suivantes nous revenons plus en détail sur les représentations à ajouter ainsi que sur les modifications apportées aux phases de MCTS présentées dans le chapitre 4.

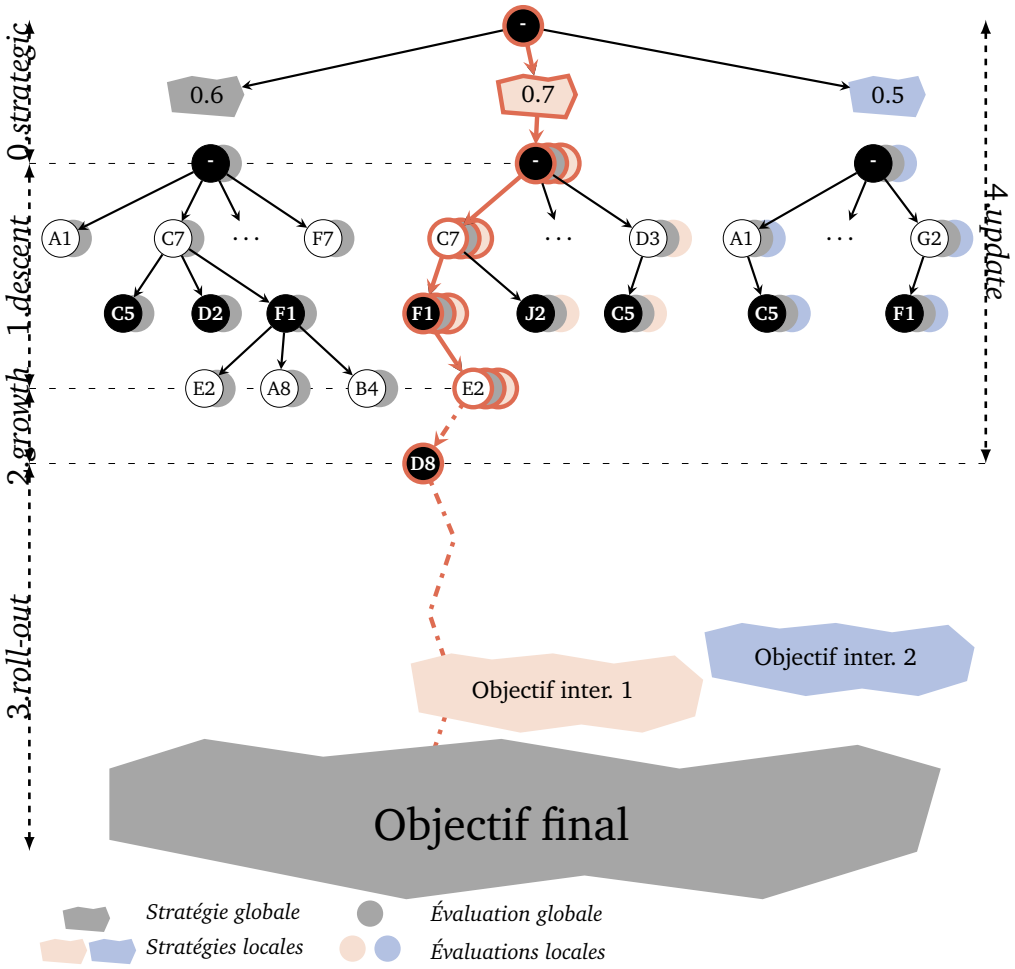


Figure 6.4.– Phases d'un épisode de MCTS dans le modèle avec stratégies

6.2.1.1. Représentations ajoutées à MCTS

Dans ce modèle, les stratégies sont évaluées dans une arborescence indépendante. Comme nous supposons qu'au maximum une seule stratégie locale sera appliquée par épisode, l'arbre sera de profondeur 1. La racine représente la situation courante et chaque noeud fils correspond à l'application d'une stratégie depuis cette position. Les

noeuds stratégiques sont évalués conformément à l'objectif final comme n'importe quelle action dans MCTS.

À chaque stratégie est associée un arbre de recherche indépendant [2]. Les mêmes états sont susceptibles d'être traités par plusieurs arbres différents mais les valeurs accumulées correspondent à des perspectives de recherche distinctes. Contrairement à l'arbre de MCTS, les noeuds des arborescences locales comportent deux évaluations :

— une évaluation globale $Q^{\pi^j}_{globale}$;

— une évaluation locale $Q^{\pi^j}_{locale}$.

L'évaluation globale $Q^{\pi^j}_{globale}$ est calculée suivant l'objectif final, comme celle habituellement calculée par MCTS. L'évaluation locale $Q^{\pi^j}_{locale}$ est calculée suivant l'objectif intermédiaire spécifique à la stratégie. Les noeuds associés à la stratégie globale ne comportent qu'une seule évaluation globale, à l'instar de l'arbre de recherche classique.

6.2.1.2. Modifications apportées aux phases de MCTS

Le déroulement d'un épisode reste semblable à celui développé dans MCTS. Au début de chaque épisode, nous introduisons simplement une nouvelle phase que nous nommerons *strategic*. Les phases de *descent* et *update* sont légèrement modifiées tandis que les phases *growth* et *roll-out* restent identiques par défaut. Il est toutefois possible d'adapter la phase de *growth* ou de *roll-out* pour chaque stratégie locale, en proposant par exemple d'initialiser les noeuds différemment lors de la phase de *growth*. Nous revenons succinctement sur les modifications à apporter à chacune des phases *strategic*, *descent* et *update*.

Phase *strategic* La phase *strategic* sélectionne la stratégie à adopter au cours de l'épisode. À chaque nouvel épisode, l'agent doit décider parmi un panel d'options, la plus prometteuse à partir des évaluations progressivement calculées. En d'autres termes, cela correspond à un problème de bandit à $n-$bras (cf. Annexe A.2). C'est pourquoi, nous avons alors choisi naturellement une politique de type UCT disposant de son propre coefficient d'exploration c_{strat} que nous appellerons par la suite coefficient d'exploration stratégique.

Phase *descent* La phase *descent* sélectionne les premières actions de l'épisode dans l'arborescence de la stratégie choisie et selon la politique correspondante. La politique de *descent* de la stratégie globale considère uniquement l'évaluation globale associée

2. Les arbres sont strictement indépendants d'un point de vue conceptuel. En mémoire, un seul et même arbre mutualise les évaluations associées aux différentes stratégies.

à ses noeuds, à l'image de la méthode MCTS classique. Les politiques de *descent* associées aux stratégies locales considèrent une combinaison entre l'évaluation locale et l'évaluation globale pour décider de la prochaine action [3]. Dans le cadre de jeux à deux joueurs, cette combinaison n'est appliquée que pour les actions du joueur courant : les actions de l'adversaire sont choisies à partir de l'évaluation globale uniquement. Cette politique est appliquée jusqu'à ce qu'un état en dehors de l'arbre soit sélectionné ou bien lorsque la condition de terminaison est vérifiée. Le cas échéant, la phase de *descent* se poursuit par défaut dans l'arborescence de la stratégie globale avec la politique de *descent* de cette même stratégie comme cela est présenté dans la figure 6.5. Dès qu'un état en dehors de l'arbre est atteint, les phases de *growth roll-out* prennent le relais. L'ensemble des techniques habituellement employées dans les politiques de *descent* peuvent être adaptées pour des stratégies locales comme par exemple UCT ou RAVE (cf. section 4.2.1.1).

Phase *update* La phase *update* renforce les actions sélectionnées lors de la phase de *descent*, une fois l'épisode terminé. Depuis la position final, le résultat global est calculé comme dans la méthode MCTS. D'autres résultats spécifiques sont susceptibles d'être calculés lorsqu'une stratégie locale a été adoptée [4]. Ensuite les noeuds sont renforcés conformément à la politique de *descent* avec laquelle ils ont été choisis. Si la stratégie globale a été choisie, seul les noeuds de l'arborescence sont renforcés à partir du résultat final. Si une stratégie locale a été appliquée, les noeuds correspondant à cette arborescence sont renforcés à partir du résultat final et du résultat spécifique. Si jamais la stratégie globale a succédé à l'application d'une stratégie locale, les noeuds sont renforcés différemment suivant l'arborescence dans laquelle ils se trouvent, comme cela est présenté dans la figure 6.6.

6.2.2 Réification d'objectifs intermédiaires dans MCTS

Les stratégies locales du modèle présenté suivent chacune un objectif intermédiaire différent. La dynamique d'apprentissage suivant un objectif ne pose *a priori* aucun problème. Cependant la mise en oeuvre des dynamiques d'apprentissage locales posent plusieurs problèmes dans leur mise oeuvre. Tout d'abord, nous nous intéressons à atteindre une configuration pertinente pour la stratégie locale sans définir de consignes *a priori* pour sa réalisation. Dans la section 6.2.2.1, nous précisons comment matérialiser cet objectif au travers d'une rétroaction. Ensuite ces objectifs intermédiaires sont conçus initialement pour servir un objectif final essentiel : gagner la partie. Nous présentons comment concilier ces deux objectifs au cours de l'apprentissage, dans la section 6.2.2.2.

3. Nous reviendrons plus longuement sur le calcul précis de cette valeur dans la section 6.2.2.2.
4. Nous reviendrons plus longuement sur le calcul de ce résultat spécifique dans la section 6.2.2.1.

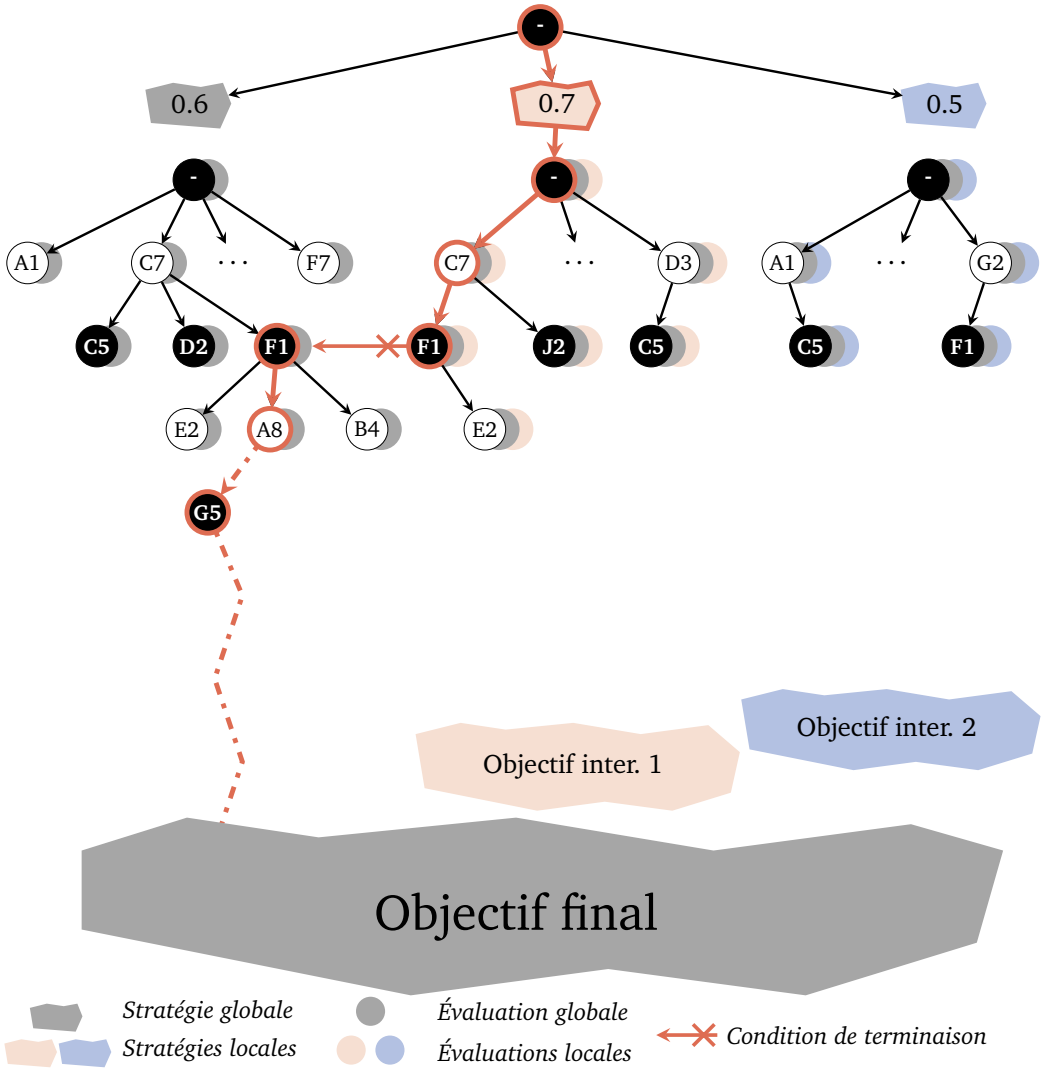


Figure 6.5.– Parcours de l'arbre de recherche selon des stratégies locales lorsque la condition de terminaison est atteinte.

6.2.2.1. Rétroaction pour un objectif intermédiaire

La rétroaction représente la valeur de référence qui sert à ajuster progressivement le comportement de l'agent (cf. section 2.2.2). À la différence de l'objectif final, les objectifs intermédiaires ne disposent pas d'une rétroaction immédiate comme le résultat d'une partie. La définition de la rétroaction pour une stratégie locale est à préciser suivant sa mise en application. À défaut d'autres références, c'est essentiellement au travers de la rétroaction que l'objectif intermédiaire sera défini.

Les rétroactions complémentaires associées aux objectifs intermédiaires sont obtenues à partir de résultats calculés en fin d'épisode, à l'instar de la *Criticality* (cf.

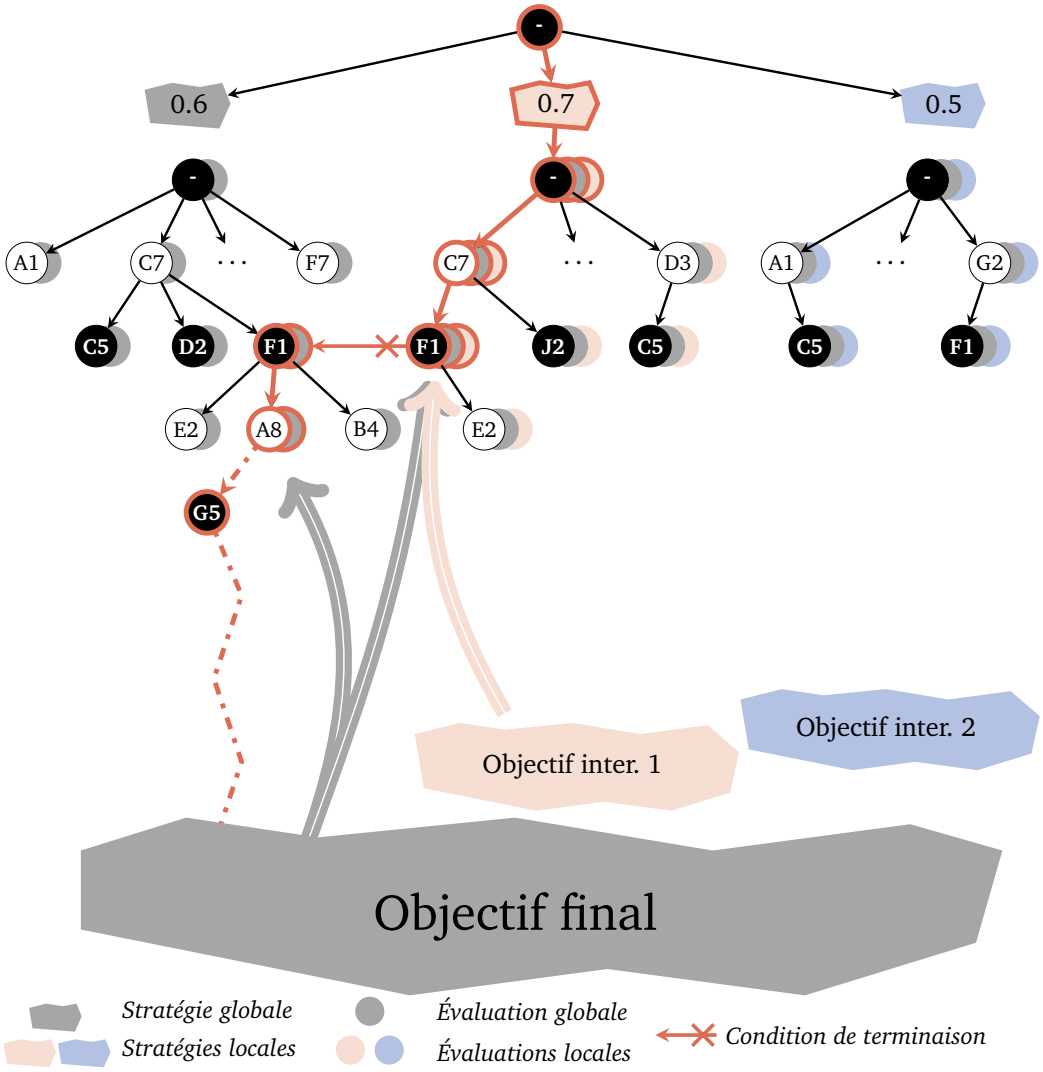


Figure 6.6.– Mise à jour de l'arbre de recherche suite à l'application d'une stratégie locale, puis d'une stratégie globale.

section 4.2.1.4, ou bien à partir d'informations récupérées au cours de la simulation [5]. Par exemple, pour établir l'objectif « capturer le groupe de pierres blanches du bas », dans la figure 6.7, il suffit d'observer en fin d'épisode si la zone coloriée en rouge est toujours possédée exclusivement par Blanc. Bien évidemment, il s'agit d'un exemple purement illustratif car un tel objectif nécessite de pouvoir identifier les différents groupes et les zones à considérer à la volée.

6.2.2.2. Équilibre entre objectif final et objectif intermédiaire

La démarche adoptée pour ce modèle se distingue d'une approche *divide-and-conquer* car les objectifs intermédiaires ne doivent pas être résolus indépendamment mais

5. À la différence de l'objectif final, certains objectifs intermédiaires sont susceptibles d'être atteints au cours de l'épisode.

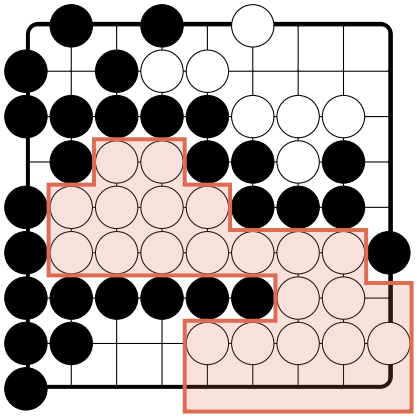


Figure 6.7.– Calcul d'une rétroaction pour le jeu de Go

doivent contribuer à la réalisation d'un objectif final. Cette problématique se retrouve aussi bien dans la problématique de *descent* dédiée que dans la condition de terminaison de la stratégie ; c'est à dire : à quel moment interrompre la stratégie locale ?

Politique de *descent* Bien qu'il soit en principe nécessaire, l'objectif intermédiaire ne doit pas être suivi au détriment de l'objectif final. La politique de *descent* doit habilement concilier la satisfaction des deux objectifs au travers des valeurs locales et globales. L'équilibre entre objectif intermédiaire et objectif final correspond à un problème d'optimisation à deux objectifs [6]. L'approche la plus immédiate consiste à fusionner les deux valeurs en une combinaison linéaire comme présenté dans l'équation 6.1. L'équilibre entre l'évaluation locale et l'évaluation globale est paramétré par le coefficient ω que nous appellerons par la suite coefficient stratégique.

$$Q^{\pi_j}(s,a) = \omega.Q^{\pi_j}_{locale}(s,a) + (1-\omega).Q^{\pi_j}_{globale}(s,a) \quad , \tag{6.1}$$

tel que

$Q^{\pi_j}_{locale}$: valeur locale pour la stratégie π_j
$Q^{\pi_j}_{globale}$: valeur globale pour la stratégie π_j
ω : coefficient stratégique compris entre 0 et 1

Des méthodes d'apprentissage par renforcement multi-objectifs proposent par exemple de considérer d'autres méthodes de calcul que la combinaison linéaire, à l'instar, par exemple, de la norme de Chebyshev [DN13] ou bien des indicateurs d'hypervolume [WS12]. Cependant, par soucis de simplification, nous faisons le choix

6. Il s'agit toutefois d'un problème particulier d'optimisation multi-objectifs car un objectif final dépend de l'objectif intermédiaire

d'exploiter les propriétés de la combinaison linéaire, dans un premier temps. Il est toutefois possible que certaines méthodes de calcul soient préférables à d'autres suivant la rétroaction définie.

Condition de terminaison Indépendamment du calcul de la politique de *descent*, il n'est pas toujours évident de concevoir des objectifs intermédiaires qui représentent des points de passage obligés pour atteindre l'objectif final. Les objectifs intermédiaires peuvent s'avérer être des points de repères (*landmark* en anglais) vers lesquels se diriger, sans pour autant les atteindre nécessairement. Dans un tel cas de figure, la condition de terminaison est décisive du succès de la stratégie locale. Par exemple, il n'est généralement pas nécessaire, dans une partie de Go, de capturer un groupe de pierres à tout prix mais il suffit parfois de le circonscrire suffisamment pour satisfaire d'autres objectifs. L'approche la plus immédiate consiste à définir un seuil tel que, l'objectif intermédiaire est considéré comme atteint lorsque la valeur locale atteint ce seuil.

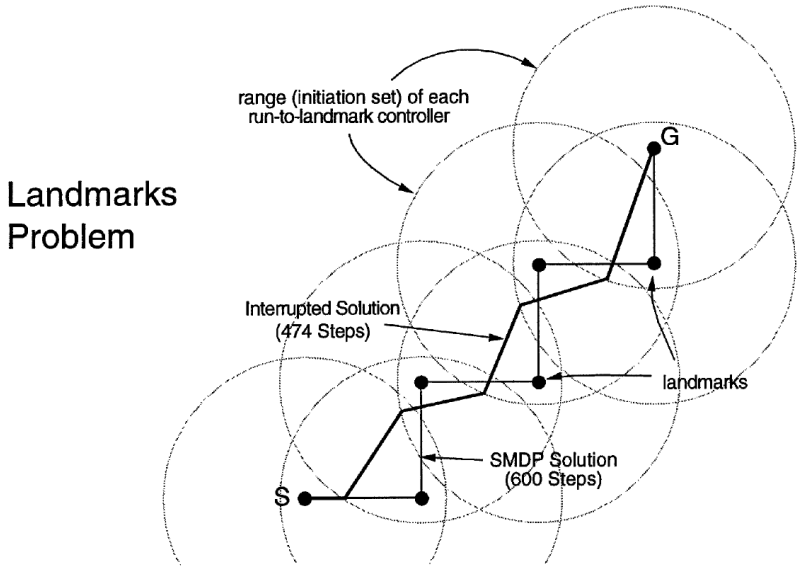


Figure 6.8.– Illustration des *Interrupting options* dans un problème de navigation [7]. Dans ce problème, l'agent doit apprendre à se déplacer du point S au point G. 5 objectifs intermédiaires lui ont été fournis au préalable en guise de points de repères. La méthode sans *Interrupting options* rejoint systématiquement les points de repères avant d'atteindre le suivant. La méthode avec les *Interrupting options* délaisse la stratégie locale avant d'atteindre l'objectif intermédiaire, permettant ainsi d'atteindre plus rapidement l'objectif final.

Dans leur article sur les PDSM[SPS99], Sutton, Singh et Precup proposent un autre mécanisme reposant sur l'évaluation globale de la stratégie : les *Interrupting options* (« options qui s'interrompent » en français). Une illustration de ce mécanisme est proposé en figure 6.8. Dans le modèle que nous proposons, les *Interrupting options*

7. Cette image est extraite de l'article [SPS99] et est reproduite ici avec l'autorisation d'Elsevier.

s'expriment de la façon suivante : la stratégie locale s'interrompt pour un état donné dès que la valeur globale, en suivant une stratégie locale, est inférieure à la valeur globale en suivant la stratégie globale. En d'autres termes, la poursuite de la stratégie locale s'interrompt dès que son approfondissement est moins bon que si aucune stratégie n'était appliquée. Dans notre situation ce mécanisme semble difficile à mettre en oeuvre. Cela signifierait que tout état couvert par l'une des stratégies locales devrait être aussi couvert par l'arborescence de la stratégie globale : ce qui n'est pas garanti pour le modèle proposé et fait également perdre de « l'autonomie de recherche » à la stratégie, ce qui n'est pas le but recherché. Néanmoins ce mécanisme illustre bien l'intérêt de la condition de terminaison d'une stratégie locale.

6.2.3 Extension du modèle proposé

Le modèle proposé pose les fondements d'une méthode MCTS avec plusieurs stratégies locales, cependant de nombreuses extensions à ce modèle sont envisageables par la suite. Nous présentons ici celles qui nous paraissent les plus intéressantes au regard du présent manuscrit.

6.2.3.1. Mutualisation de la connaissance

Pour cette extension nous nous intéressons aux partages possibles entre les différentes explorations dédiées à chacune des stratégies. Le modèle présenté se base sur la méthode MCTS. Or l'arbre de recherche utilisé par MCTS est une représentation sujette à une forme de redondance de l'information (cf. Définition 3.9 à la page p. 36). Les arborescences entre les stratégies sont distinctes. Par conséquent le même état doit être exploré systématiquement par chacune des stratégies. Les renforcements au cours d'un même épisode bénéficient exclusivement à la stratégie, au risque d'apprendre les mêmes informations pour différentes stratégies voire d'en oublier une grande partie d'un tour sur l'autre. Nous avons développé, au cours de ce manuscrit, deux approches différentes pour limiter cette redondance d'information.

La première consiste à distribuer la même rétroaction à un plus grand nombre d'états à l'instar de RAVE (cf. section 4.2.1.4). Le mécanisme *intra-option learning* proposé par Sutton Precup et Singh[SPS99] propose en l'occurrence de partager les mêmes renforcements à plusieurs stratégies différentes, en comparant les actions proposées par chacune des stratégies pour ce même état. Si jamais l'action qui aurait été appliquée par une autre stratégie était la même que celle effectivement choisie, alors l'action serait aussi renforcée pour cette stratégie.

La seconde consiste à proposer une représentation plus adéquate pour mémoriser les actions jouées. En effet, il est raisonnable de penser que l'accomplissement

d'une stratégie locale ne dépend pas systématiquement de la totalité de l'état. Des représentations moins précises sous la forme d'un arbre local, seraient plus à même d'apprendre à résoudre une stratégie locale, comme par exemple l'arbre direct de schémas présenté dans le chapitre 5.

6.2.3.2. Stratégies locales enchaînées

Nous nous sommes ici restreint à une seule stratégie locale par épisode au maximum : l'arborescence spécifique aux stratégies est limitée à une profondeur de 1. Ce choix a été réalisé pour observer, dans un premier temps, le comportement de MCTS avec une seule stratégie locale. À terme, il serait envisageable de considérer une arborescence supérieure à 1 pour les stratégies. L'agent serait en mesure de considérer plusieurs objectifs intermédiaires successifs avant d'atteindre l'objectif final. En d'autres termes, la stratégie globale ne serait plus systématiquement adoptée lorsque la condition de terminaison est vérifiée pour une stratégie locale : une autre stratégie locale prendrait éventuellement la suite de la phase de *descent*.

La démarche la plus naturelle consiste à associer, à chaque noeud de l'arborescence stratégique, l'état à partir duquel la stratégie a été initiée. Une arborescence parallèle à celle des états couverts par MCTS serait constituée et serait consultée lorsqu'une stratégie locale prend fin. Les stratégies sont alors comparées selon leurs valeurs globales respectives comme au niveau de la racine [8]. Cependant cette démarche se heurte à la limitation intrinsèque de la représentation sous forme d'un arbre de recherche : redondance et perte d'information (cf. Définition 3.9 à la page p. 3.9). Une autre démarche consiste à caractériser les états initiaux pour chacune des stratégies à partir de représentations partielles et ainsi de généraliser les noeuds stratégiques à un plus grand nombre d'états.

6.2.3.3. Construction dynamique de l'estimateur d'une stratégie locale

Dans le modèle proposé, aussi bien le nombre d'objectifs que les zones utilisées pour calculer la rétroaction, sont définis *a priori*. Ainsi le modèle n'est pas en mesure d'établir à la volée des stratégies associées aux différents groupes de pierres, comme par exemple ceux présentés dans la figure 6.7. Il faudrait que l'agent ait la capacité d'identifier les différents groupes de pierres ainsi que leur zone d'influence.

8. En pratique, cette arborescence stratégique se construit de fait dans les arborescences de chacune des stratégies puisque les noeuds couverts portent la valeur globale associée à leur stratégie, pour cet état.

Une telle extension est envisageable en reprenant par exemple les méthodes employées par les programmes de Go avant l'avènement de MCTS, ou bien des méthodes reposant sur des simulations de Monte Carlo comme celle proposée par Graf et Platzner pour identifier à la volée des zones du goban particulièrement critiques [GSP14].

6.3 Application aux jeux de Go et Clobber Solitaire Impartial

MCTS est une méthode applicable en principe à n'importe quel jeu. L'intégration de stratégies locales suivant le modèle exposé au cours du chapitre précédent ne remet pas en cause sa généricité. Les expérimentations présentées ici servent à étudier le comportement de MCTS avec le modèle de stratégies locales proposé pour différents jeux : les jeux de Go et du Clobber Solitaire Impartial (CSI) (cf. section 2.1). Dans la section 6.3.1 nous évoquons les paramètres généraux du modèle à l'étude ainsi que les indicateurs de performances retenus au cours des expériences.

Toutefois certains aspects déterminants du modèle restent à préciser selon le jeu retenu, comme par exemple le choix et la mise en oeuvre des objectifs intermédiaires. Pour chacun de ces jeux, l'influence des rétroactions et l'équilibre entre l'objectif intermédiaire et l'objectif final n'auront pas la même portée. Dans les sections 6.3.2 et 6.3.3 nous précisons les mises en oeuvre du modèle pour chacun de ces jeux ainsi que les résultats expérimentaux obtenus. Les résultats seront ensuite plus longuement interprétés dans la section 6.3.4.

Comme le suggère la section 6.2.3, de multiples extensions sont réalisables. Il s'agit dans un premier temps d'observer la réaction du modèle afin de dégager des perspectives d'amélioration.

6.3.1 Protocole expérimental

Au cours de ces expérimentations, nous avons souhaité étudier le comportement du modèle général pour différentes mises en application. Dans un premier temps, nous détaillons les principaux paramètres communs aux deux jeux, que nous allons faire varier au cours des expériences. Les aspects spécifiques à chacun des jeux, tant au niveau du modèle qu'au niveau de l'implémentation, seront précisés dans les parties correspondantes. Nous revenons ensuite brièvement sur les indicateurs de performances retenus pour les expérimentations.

6.3.1.1. Paramètres

Le modèle présenté en section 6.2 étend la méthode MCTS en introduisant de nouveaux paramètres. Au cours de ces expérimentations nous allons faire évoluer quatre paramètres différents. Les deux premiers, la taille du plateau et le nombre de simulations servent ici à tester les capacités du modèle lorsque la difficulté du problème augmente ou bien lorsque le modèle dispose d'une plus grande puissance de calcul. Les deux autres paramètres, c_{strat} et ω, représentent les deux variables du modèle commun que nous allons plus longuement tester pour le jeu du CSI. L'influence de chacun d'entre eux est détaillée plus longuement ci-dessous :

— La taille du plateau △ : `8x8` à `16x16` pour le CSI et 9x9 pour le Go : induit une complexité plus grande pour MCTS selon la taille du plateau. En effet le facteur de branchement en début de partie sera d'autant plus grand pour un grand plateau ;

— Le nombre de simulations ▼ : `1000` à `32000` : indique le nombre de simulations autorisées aux programmes avant de jouer la prochaine action. Un plus grand nombre de simulations offre la possibilité à l'agent d'apprendre une plus grande quantité d'informations et, en principe, d'améliorer sa capacité de jeu ;

— Le coefficient d'exploration stratégique c_{strat} ■ :$c_{strat} \geq 0$: contrôle le compromis entre exploration et exploitation dans la formule UCT pour le choix des stratégies (cf. section 6.2.1.2). De la même manière, une valeur proche de 0 favorise les meilleures stratégies et une valeur plus grande laisse l'opportunité d'essayer d'autres stratégies ;

— Le coefficient stratégique ω □ :$\omega \in [0; 1]$ régule l'influence de l'objectif intermédiaire sur l'objectif final dans l'évaluation associée à la stratégie locale (cf. section 6.2.2.2). Une valeur proche de 0 privilégie l'objectif final tandis qu'une valeur proche de 1 favorise la poursuite de l'objectif intermédiaire sur l'objectif final.

6.3.1.2. Indicateurs de performances

Le travail réalisé ici vise à étudier le comportement du modèle dans un premier temps, avant de l'approfondir pour obtenir des résultats plus compétitifs. C'est pourquoi tous les résultats ont été produits pour un nombre constant de simulations par tour, et non à temps constant comme c'est l'habitude pour le jeu de Go. En effet nous en sommes encore à un étape prospective dans l'élaboration du modèle et de nombreuses optimisations sont encore réalisables.

En outre, nous nous intéressons ici plus aux tendances du programme qu'à des performances occasionnelles, même pour des problèmes d'optimisations comme c'est le cas du jeu du CSI [9]. Les statistiques proposées ont été calculées à partir de tailles conséquentes d'échantillons allant de 400, pour les configurations les plus difficiles, à 5000 parties. À chaque fois les résultats moyens seront fournis avec un indicateur de sa fiabilité : un intervalle de confiance à 95% pour un jeu à deux joueurs comme le jeu de Go, et l'écart type dans le cadre du CSI.

En plus de la performance moyenne, des indicateurs propres à MCTS seront aussi calculés pour rendre compte de sa dynamique interne. Nous observerons notamment la profondeur de la recherche atteinte pour le jeu du CSI afin d'observer la capacité du programme à concentrer sa recherche au travers du modèle de stratégies locales proposé.

6.3.2 Jeu de Go

Le jeu de Go représente un problème pour lequel l'adaptation de la dynamique de recherche est prometteuse. Au cours de la partie, seulement certaines zones du plateau requièrent une plus grande attention suivant la situation. Une dynamique d'apprentissage adaptée permettrait de rapidement écarter les actions moins pertinentes pour la situation actuelle, au profit des zones plus décisives dans l'espace de recherche. En particulier sur un goban 19x19, la combinatoire est si prégnante que la recherche gagnerait certainement à être structurée autour d'objectifs intermédiaires à l'image des précédents programmes de jeu, comme nous l'avons précédemment évoqué dans la section 6.1.2.

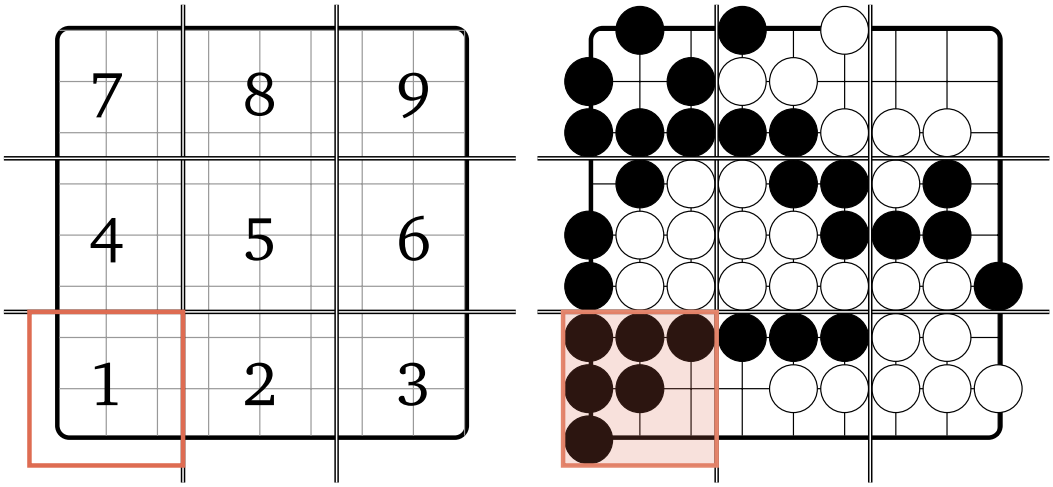
Cependant la conception d'objectifs intermédiaires pour ce jeu nécessite un développement plus approfondi spécifiquement sur cette question, que nous n'avons pas souhaité traiter dans un premier temps. Nous proposons ici une étude préliminaire sur un plateau 9x9 pour lequel des stratégies locales plus naïves sont envisageables. Dans un premier temps nous exposons brièvement les stratégies locales proposées pour cette taille de plateau. Puis nous présentons le résultat obtenu pour ce jeu.

6.3.2.1. Modèle applicatif

L'objectif final dans le jeu de Go consiste à contrôler un territoire plus grand que celui de son adversaire. Suivant cette perspective, nous avons établi un panel d'objectifs intermédiaires naïfs consistant à contrôler, pour chacun, une partie différente du

9. Le meilleur score obtenu sera néanmoins mentionné pour les résultats du CSI à titre purement indicatif.

goban [10]. Un goban de taille 9x9 est divisé en 9 zones distinctes de 3x3 intersections comme cela est présenté dans la figure 6.9a. Nous avons au total 10 stratégies différentes : la stratégie globale et une stratégie locale pour chacune de ces zones.



(a)Découpe du plateau **(b)**Calcul d'une rétroaction

Figure 6.9.– Stratégies locales pour le jeu de Go : le goban 9x9 est découpé en 9 régions de 3x3 intersections correspondant à une stratégie locale.

Rétroaction pour les objectifs intermédiaires Afin d'évaluer les actions de l'agent au regard de ces objectifs intermédiaires, nous avons défini une rétroaction associée à chaque stratégie locale. Lorsqu'une stratégie locale a été choisie, l'occupation de la zone correspondante à cette stratégie est calculée en fin de simulation. Une valeur égale à 0 signifie que l'adversaire contrôle totalement la zone en question et, à l'inverse, une valeur de 1 indique un contrôle total pour l'agent [11]. Par exemple dans la figure 6.9b, la valeur associée à la stratégie locale 1 serait de 0.83 pour le joueur Noir : 6 pierres noires et trois intersections vides ramenées à une valeur comprise entre 0 et 1. Cette valeur est ensuite utilisée pour renforcer les évaluations locales associées, lors de la phase *update*.

À la différence du modèle présenté dans la section 6.2, tous les résultats finaux sont ici mutualisés dans une arborescence commune à toutes les stratégies. Ainsi chaque état couvert par la méthode, dispose d'une unique valeur globale qui est le résultat de l'application de différentes stratégies. Cette différence introduit nécessairement un biais dans l'évaluation globale mais permet de limiter la redondance d'information due aux différentes arborescences. Les résultats préliminaires obtenus pour ce jeu confirmaient ce phénomène par des performances bien moindres pour le nombre de simulations traités ici.

10. Ces objectifs intermédiaires sont naïfs car contrôler le territoire à un instant donné au Go ne garantit nullement de le détenir en fin de parties, à cause des éventuelles captures.
11. Une zone complètement vide compte pour 0.5.

Équilibre entre objectif final et objectif intermédiaire Lorsqu'une stratégie locale est choisie, l'équilibre entre objectif final et intermédiaire est assuré lors de la sélection d'actions par une combinaison linéaire entre l'évaluation globale (et donc commune à toutes les stratégies) et l'évaluation locale spécifique à cette stratégie, comme cela est présenté dans l'équation (6.1). Toutefois la poursuite irrémédiable d'un contrôle total de la région en question, n'est pas nécessairement productive, surtout en début de partie. C'est pourquoi nous avons établi un seuil à partir duquel la stratégie locale s'interrompt au profit de la stratégie globale. En l'occurrence, la stratégie locale se termine dès que l'évaluation locale, calculée pour l'état courant, est supérieure à $7/9$. En d'autres termes, l'objectif intermédiaire est considéré comme rempli dès que 7 des 9 intersections de la zone sont contrôlées en fin de partie, en moyenne.

6.3.2.2. Résultats

Le modèle applicatif proposé ci-dessus a été implémenté à partir du programme *Fuego* dans sa version SVN (révision 1713) [12]. Ce programme de Go open-source utilise MCTS complété des améliorations classiques de la littérature : UCT, RAVE, politique de *roll-out Sequence-like*. Les résultats pour ce jeu ont été obtenus en opposant le programme *Fuego* avec le modèle de stratégies locales au même programme sans les stratégies locales.

La figure 6.10 trace l'évolution des performances pour un nombre croissant de simulations par tour pour les deux programmes. Les paramètres spécifiques au modèle proposé sont maintenus constants, pour toutes ces expérimentations, aux valeurs $c_{strat} = 1.4$ et $\omega = 0.3$. La valeur $c_{strat} = 1.4$ correspond à la valeur classique pour les problèmes de bandit à $n-$bras simple (à un seul niveau). La valeur $\omega = 0.3$ a été établie empiriquement. Nous constatons que pour des nombres relativement faibles de simulations par tour, le programme disposant de stratégies locales est légèrement meilleur que celui de référence. Ces résultats seront plus longuement analysés dans la section 6.3.4.

6.3.3 Jeu du Clobber Solitaire Impartial

Le jeu du Clobber Solitaire Impartial est un jeu pour lequel la résolution est simplifiée par rapport à des jeux à deux joueurs comme le Go (cf. section 2.1.2). En effet, le programme n'a pas à envisager les actions possibles de son adversaire. Il s'efforce de minimiser le nombre de pierres restantes sur le plateau à la fin. Par ailleurs, nous

12. La version SVN de Fuego est plus à jour que la version 1.1 utilisée dans le chapitre précédent et intègre de nouvelles améliorations pour la politique de *descent* comme celle proposée par Rosin [Ros11b].

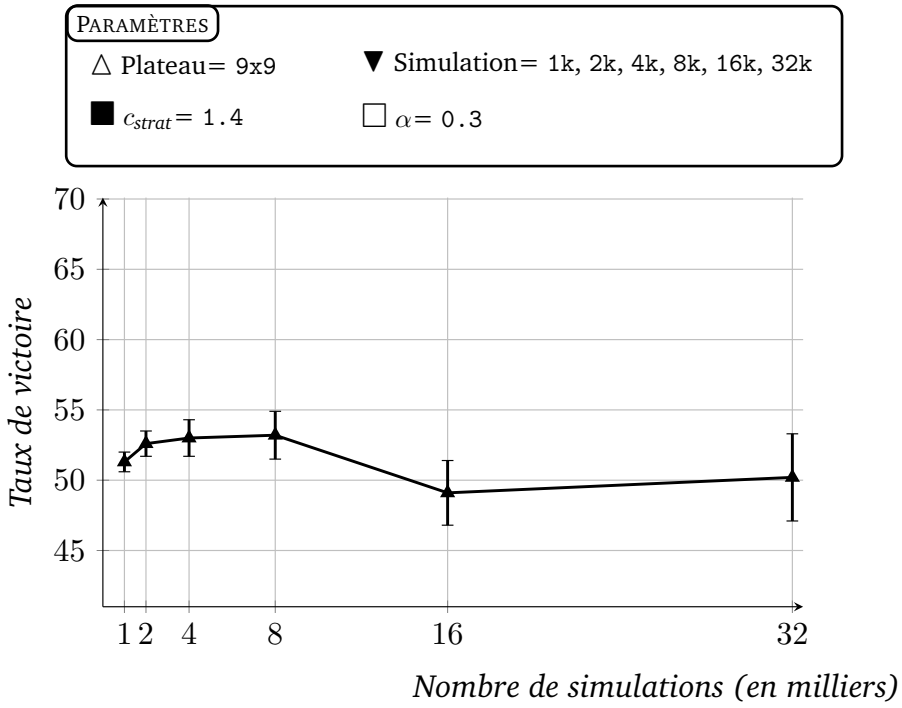


Figure 6.10.– Gain obtenu avec l'ajout de stratégies locales pour un nombre croissant de simulations par tour.

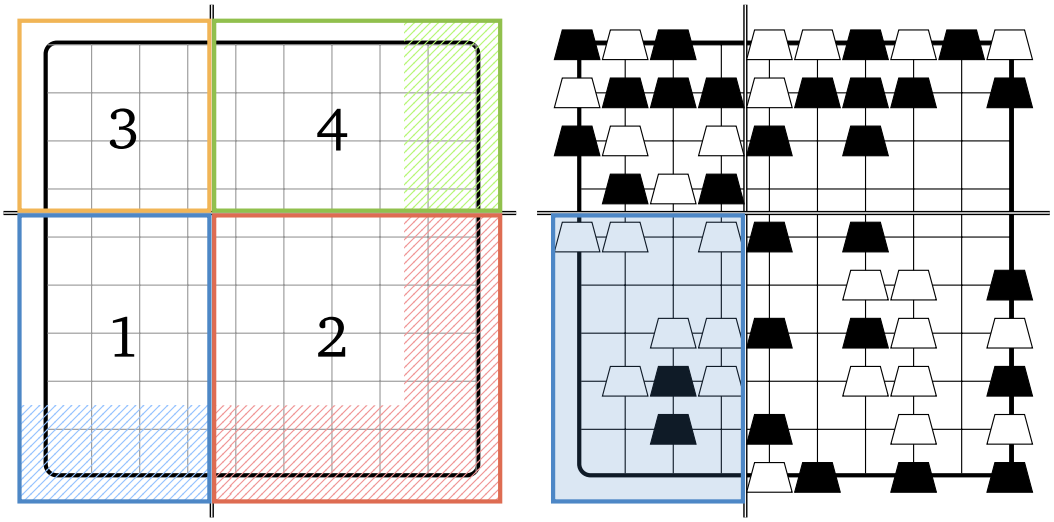
pourrons confronter le score obtenu par le joueur artificiel avec les meilleurs scores établis théoriquement (cf. Annexe A.1).

Ce jeu reste toutefois difficile pour une méthode comme MCTS. Dans une configuration initiale en damier, le nombre d'actions possibles dans les premiers tours est très élevé. Cette forte combinatoire rend le jeu très difficile à traiter par MCTS. À défaut d'informations complémentaires, la méthode disperse sa recherche sur l'ensemble de la grille au détriment d'une recherche moins profonde. Ce faisant, le joueur artificiel risque de mal évaluer certaines actions critiques et de s'éloigner irrémédiablement du meilleur score théorique. Les stratégies locales, pour ce jeu, servent en l'occurrence à approfondir momentanément la recherche autour d'objectifs intermédiaires afin de négocier correctement ces actions critiques.

6.3.3.1. Modèle applicatif

L'objectif final dans le jeu du CSI consiste à réduire au maximum le nombre de pierres restantes sur l'ensemble du plateau. Suivant cette perspective, la réduction de sous-ensembles du plateau forme un panel d'objectifs intermédiaires idéal pour le jeu du CSI. D'ailleurs, la démonstration proposée par Beaudou, Duchêne et Gravier pour un plateau en forme de grille, procède ainsi [BDG13]. Le plateau est découpé en sous-ensembles d'au moins 4x4 intersections ou bien légèrement plus grand dans

le cas de grilles dont la taille n'est pas un multiple de 4. Ils ont alors montré qu'il est possible de vider successivement chacune de ces régions pour finir avec au plus 2 pierres dans la dernière région. Nous nous sommes inspirés de la découpe proposée par cette démonstration pour définir le périmètre des différents sous-ensembles comme cela est présenté dans la figure 6.11a. En plus de la stratégie globale, une stratégie locale est dévolue à la réduction de chacune des régions définies.



(a)Découpe du plateau **(b)**Calcul d'une rétroaction

Figure 6.11.– Stratégies locales pour le jeu de Clobber Solitaire Impartial : le plateau est découpé en régions de 4x4 intersections. Lorsque la taille du plateau n'est pas un multiple de 4, les régions sur le bord sont légèrement étendues par les intersections supplémentaires. À chaque région est associée une stratégie locale.

Rétroaction pour les objectifs intermédiaires Afin d'évaluer les actions de l'agent au regard de ces objectifs intermédiaires, nous avons défini une rétroaction associée à chaque stratégie locale. Lorsqu'une stratégie locale a été choisie, le taux de pierres retirées dans la zone correspondante est calculé en fin de simulation. Une valeur égale à 0 signifie que la zone est encore pleine et une valeur égale à 1 signifie à l'inverse que la zone est complètement vide. Par exemple dans la figure 6.11b, la valeur associée à la stratégie locale 1 serait de 0.375 : 9 pierres restantes pour 24 intersections au total. Cette valeur est ensuite utilisée pour renforcer les évaluations locales associées, lors de la phase *update*.

Équilibre entre objectif final et objectif intermédiaire Comme dans le modèle appliqué au jeu de Go, l'équilibre entre objectif final et intermédiaire est assuré lors de la sélection d'actions par une combinaison linéaire entre l'évaluation globale et l'évaluation locale spécifique à cette stratégie (cf. Équation (6.1)). Cependant la condition de terminaison de la stratégie locale est sensiblement différente. En effet une fois qu'une pierre se retrouve isolée des autres, il n'y a plus aucune chance qu'elle ne soit retirée du plateau par la suite. À fixer un seuil, nous courons le risque qu'une

stratégie locale ne se termine jamais à cause d'un nombre trop grand de pierres isolées. Dans le modèle appliqué ici, la stratégie locale s'interrompt lorsque l'évaluation locale estimée par simulations de Monte Carlo est environ égale à l'évaluation locale actuelle de l'état traversé. Ainsi, la stratégie locale s'interrompt lorsqu'il n'est plus possible d'agir sur la région, que ce soit à cause de pierres isolées ou bien tout simplement lorsque la région a été vidée.

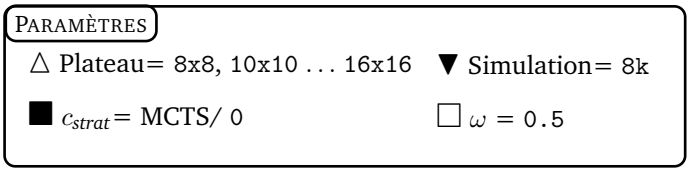
6.3.3.2. Résultats

Afin de réaliser les expérimentations pour ce jeu, nous avons simplement adapté la méthode MCTS pour pouvoir traiter des jeux solitaires qui ne comportent pas véritablement de tours de jeu. Avant chaque action, le joueur artificiel calcule en interne la meilleure action possible à partir d'un nombre fixe de simulations. Une fois le nombre de simulations atteint, la meilleure action apprise est jouée et la recherche reprend depuis le nouvel état du plateau. Ce déroulement de la partie, semblable à celui habituellement mis en oeuvre dans les jeux à deux joueurs, s'arrête ici lorsque le joueur n'a plus aucune action possible. D'autre part beaucoup plus d'informations de l'arbre de recherche sont conservées d'une recherche à l'autre, en l'absence d'un joueur contradicteur.

Comme nous l'avons évoqué précédemment, ce jeu constitue essentiellement un banc d'essai pour étudier le comportement de MCTS dans un environnement simplifié. C'est pourquoi nous n'avons pas souhaité intégrer à cette méthode des heuristiques spécifiques à un jeu solitaire comme cela est suggéré dans la méthode SP-MCTS [Sch+08] ou bien spécifiques au jeu du CSI. La politique de *roll-out*, par exemple, est purement aléatoire et la politique de *growth* considère de la même manière toutes les actions ajoutées à l'arbre. Seule l'amélioration UCT a été intégrée à la politique de *descent* afin d'assurer au minimum un compromis entre exploration et exploitation. La valeur de la constante d'exploration a été empiriquement fixée à 0.05 pour toutes les expérimentations présentées ici. Le programme de jeu a été développé à partir de la librairie open-source SmartGame qui est à la base du programme Fuego [Enz+09].

Parmi les résultats présentés pour ce jeu, nous observons tout d'abord l'impact des stratégies locales pour des problèmes de plus en plus difficiles testant ainsi sa résistance à la combinatoire. Ensuite nous étudions successivement l'influence des deux paramètres spécifiques au modèle, coefficient d'exploration stratégique et coefficient stratégique, sur les performances et le comportement du programme. Pour toutes les expérimentations réalisées, la configuration initiale du plateau est la plus difficile qui soit pour MCTS : la configuration en damier.

Résistance à la combinatoire Dans ces expériences, nous avons fait varier progressivement la taille initiale du plateau. La combinatoire du problème croît conjointement à la taille du plateau. Ainsi un plateau de taille 16x16 sera beaucoup plus difficile à résoudre par MCTS qu'un plateau 8X8. Dans la table 6.1, nous présentons les résultats obtenus par la méthode MCTS seule et par la méthode MCTS avec des stratégies locales, conjointement aux meilleures scores théoriques. Nous pouvons constater que pour toutes les tailles de plateau, la méthode avec des stratégies locales est légèrement meilleure que la méthode MCTS pour les paramètres considérés. Ce gain est plus sensible pour les plateaux 12x12 et 16x16 pour lesquels le meilleur score atteint au cours des expérimentations est, lui aussi, légèrement meilleur.



Taille	MCTS		MCTS avec strat.		Score théorique
	Moyenne	Meilleur	Moyenne	Meilleur	
8x8	6.83 ± 1.44	2	6.72 ± 1.40	3	1
10x10	11.68 ± 1.77	6	11.47 ± 1.80	6	1
12x12	17.94 ± 2.08	14	17.55 ± 2.19	12	2
14x14	25.61 ± 2.41	20	25.32 ± 2.6	20	1
16x16	35.4 ± 2.97	29	34.94 ± 3.22	27	1

Table 6.1.– Score pour différentes tailles de plateau avec 8000 simulations par tour.

Influence du coefficient d'exploration stratégique Dans ces expériences, nous étudions l'influence du coefficient d'exploration stratégique c_{strat} sur les performances du programme avec des stratégies locales. Lors de la phase *strategic*, ce coefficient sert à équilibrer l'exploitation des stratégies disposant pour le moment de la meilleure évaluation, avec l'exploration de stratégies moins bien évaluées mais susceptibles de le devenir plus tard (cf. Annexe A.2 pour de plus amples détails). Les performances atteintes par le modèle de stratégies locales pour le jeu du CSI sont regroupées dans la table 6.2 pour des valeurs croissantes de c_{strat}. D'après les résultats obtenus, le programme produit les meilleures performances pour un coefficient d'exploration nul ; en d'autres termes, la stratégie avec la meilleure évaluation courante est systématiquement sélectionnée lors de la phase *strategic*.

Influence du coefficient stratégique Dans ces expériences, nous étudions l'influence du coefficient stratégique ω sur les performances du programme mais aussi sur son

PARAMÈTRES

△ Plateau= 8x8 ▼ Simulation = 16k

■ c_{strat}= 0, 0.05, 0.2, 0.5 □ ω= 0.4

c_{strat}	Moyenne	Meilleur score	Score théorique
0	6.18 ± 1.41	3	1
0.05	6.51 ± 1.37	5	1
0.2	6.62 ± 1.29	5	1
0.5	6.61 ± 1.32	5	1

Table 6.2.– Score suivant l'équilibre établi entre exploration et exploitation au niveau des stratégies (paramètre c_{strat}).

comportement en interne. Le coefficient ω régule l'importance de l'évaluation locale sur l'évaluation globale lorsqu'une stratégie locale est appliquée. Dans la table 6.3, nous indiquons les performances obtenues pour différentes valeur de ω ainsi que pour MCTS sans stratégies, en guise de comparatif. D'après les résultats obtenus, les performances sont légèrement meilleures en moyenne pour un coefficient stratégique de 0.5. Une valeur de 0.5 signifie que l'évaluation locale est aussi importante que l'évaluation globale pour chacune des stratégies locales. Ces résultats attestent du comportement externe de l'agent mais ne restituent pas comment l'agent a structuré sa recherche en interne.

PARAMÈTRES

△ Plateau= 8x8 ▼ Simulation= 16k

■ c_{strat}= MCTS / 0 □ ω= 0.5, 0.7, 1.0

MCTS/ω	Moyenne	Meilleur score	Score théorique
MCTS	6.23 ± 1.44	2	1
0.5	6.14 ± 1.34	2	1
0.7	6.3 ± 1.41	3	1
1.0	7.43 ± 2.09	3	1

Table 6.3.– Score moyen suivant l'importance de l'évaluation locale (paramètre ω) pour 16000 simulations par tour. Ces résultats sont à mettre en correspondance avec la figure 6.12.

En plus de la performance, nous avons observé pour ces expériences un indicateur interne à l'agent : la profondeur maximale atteinte par l'arbre lors de la recherche. La figure 6.12 en retrace l'évolution au cours de la partie pour chacun des programmes étudiés dans la table 6.3. Nous pouvons constater tout d'abord que la profondeur de la recherche atteinte par la méthode MCTS sans stratégie, reste constante à 3 au cours des 20 premières actions, pour ensuite atteindre un pic de 7.7 en moyenne, au niveau de la 48ème action jouée. Pour les trois ω différentes, la profondeur maximale de la recherche s'accroît plus rapidement qu'avec la méthode MCTS dans les premiers tours, pour finalement rejoindre le même pic final autour de la 48ème action. Toutefois, l'évolution de cet indicateur varie significativement suivant la valeur ω. Pour les valeurs $\omega = 1.0$ et $\omega = 0.7$, la profondeur de la recherche forme 3 cloches distinctes au cours de la partie. Cette évolution est moins marquée pour la valeur $\omega = 0.5$ dont la profondeur maximale reste supérieure à celle de la méthode sans stratégie jusqu'à la 39ème action. Un approfondissement plus rapide de la recherche n'implique pas nécessairement de meilleures performances (cf. table 6.3) mais cela témoigne de l'influence des stratégies locales, et du coefficient ω en particulier, sur le comportement interne de l'agent.

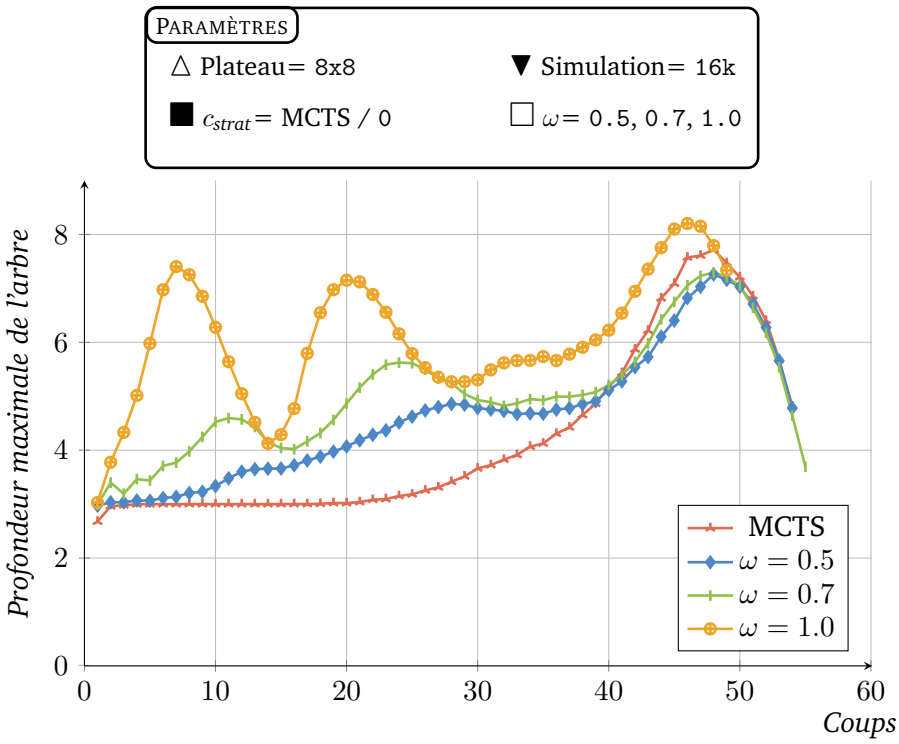


Figure 6.12.– Profondeur maximale moyenne suivant l'importance de l'évaluation locale (paramètre ω). Ces résultats sont à mettre en correspondance avec la table 6.3.

6.3.4 Interprétation des résultats

Dans cette section, nous analysons les résultats présentés pour les deux jeux. Dans un premier temps, nous revenons sur les gains applicatifs des performances observées lors de la résolution de chacun de ces jeux. Nous nous intéressons ensuite à l'influence des stratégies locales sur le comportement interne de l'agent au travers notamment d'indicateurs spécifiques à MCTS, comme la profondeur maximale de l'arbre de recherche. Ces résultats s'inscrivent davantage dans une perspective d'interprétation cognitiviste du programme. Enfin nous discutons des différents verrous posés au modèle actuel et dégageons différentes perspectives d'amélioration.

6.3.4.1. Performance du modèle avec stratégies locales

Au cours de ces expérimentations, nous avons tout d'abord évalué l'influence des stratégies locales sur les performances de MCTS pour les jeux du Go et du CSI. Ces influences sont mesurées au travers de la figure 6.10 et de la table 6.1 pour les jeux de Go et du CSI respectivement. Le cadre de définition de la proposition présentée est générique et présente de nombreuses pistes d'amélioration (cf. discussion en section 6.3.4.3). Cependant, pour ces deux jeux, un premier paramétrage permet un léger gain de performance par rapport à la méthode sans stratégies locales.

Dans les modèles applicatifs proposés, nous nous sommes efforcés d'introduire le minimum de connaissances spécifiques au jeu concerné. Pour le Go par exemple, la découpe du plateau est très arbitraire. De même, pour le jeu du CSI, les stratégies locales distinguent les différentes actions à partir des seules évaluations apprises : il aurait été par exemple envisageable d'adapter la politique de *growth* afin de privilégier les actions de la zone correspondante. Nous souhaitions ici juger du modèle commun et non d'une mise en application spécifique. C'est pourquoi cette influence, bien que ténue, nous encourage à approfondir ce modèle afin d'accentuer encore le gain de performance observé.

6.3.4.2. Influence des stratégies locales sur le comportement de MCTS

Pour le jeu du CSI en particulier, nous nous sommes intéressé à l'incidence du modèle sur le comportement du programme. Les résultats obtenus pour les paramètres ω et c_{strat} nous permettent d'observer dans quelle mesure le modèle proposé répond aux attentes évoquées dans la section 6.1 :

— concentrer la recherche autour d'objectifs intermédiaires ;

— adapter la dynamique d'apprentissage suivant la situation.

Concentrer la recherche autour d'objectifs intermédiaires L'étude menée autour du coefficient stratégique ω atteste que le modèle présenté est capable de structurer sa recherche. En particulier, l'évolution de la profondeur maximale de l'arbre présenté dans la figure 6.12 témoigne de la concentration de la recherche. En effet, un arbre de recherche plus profond signifie que la recherche s'est restreinte à explorer un nombre limité d'actions qui lui semblaient plus intéressantes. Dans la méthode MCTS par défaut, la profondeur reste à 2 lors des 20 premiers tours. Cela signifie notamment que l'agent ne distingue pas vraiment les actions entre elles, en suivant une stratégie exclusivement globale à ce stade de la partie : la recherche se disperse dans le grand nombre d'actions possibles. La profondeur croît ensuite progressivement lorsque le nombre d'actions restantes est suffisamment faible pour pouvoir explorer les niveaux suivants. Les méthodes avec stratégies locales approfondissent toutes plus rapidement leur recherche dans les premiers tours. Cela signifie que grâce aux seules évaluations locales, le programme a réussi à en écarter virtuellement certaines afin d'approfondir localement la recherche.

Par ailleurs, les cloches formées par les courbes témoignent d'une concentration de la recherche autour d'objectifs intermédiaires sur plusieurs tours. Ce phénomène s'observe particulièrement bien pour la valeur $\omega = 1.0$. La phase ascendante correspond à l'approfondissement de la recherche autour de l'objectif intermédiaire. Cependant si jamais la condition de terminaison de la stratégie locale est atteinte, la stratégie globale prend le relais avec la dispersion que nous avons observée pour la méthode MCTS. Ainsi le maximum atteint par la courbe indique lorsque l'agent identifie dans son arbre de recherche une séquence d'actions vérifiant la condition de terminaison de la stratégie ; c'est à dire,en l'occurrence, qu'il vide au maximum la zone correspondante. La phase descendante représente l'application de la séquence identifiée plus tôt dans l'arbre de recherche, jusqu'à son terme. Par exemple pour la valeur $\omega = 1.0$, la première cloche atteint une profondeur maximale de 7.4 en moyenne après avoir joué 7 actions ; c'est à dire une séquence de 14 actions au total depuis le début de la partie. Or dans un plateau 8x8 chaque région de 4x4 se vide en 16 actions au maximum. Cela signifie qu'au cours des 7 premières actions de la partie en moyenne, l'agent a certainement mal négocié des actions critiques pour cette zone et n'arrive pas à complètement vider cette zone.

Adapter la dynamique d'apprentissage Dans le modèle proposé, la phase *strategic* sélectionne la stratégie à appliquer au cours de l'épisode au travers d'un problème de bandit à $n-$bras sur l'ensemble des stratégies, locales et globale. C'est par cette méta-dynamique d'apprentissage que la méthode adapte à la volée la dynamique d'apprentissage de l'agent. Les résultats de la figure 6.12 discutés dans le paragraphe précédent nous indiquent tout d'abord que le dynamique d'apprentissage de l'agent évolue au cours de la partie. Comme nous l'avons évoqué précédemment, la profondeur maximale de l'arbre de recherche nous indique que les stratégies locales

sont appliquées dès les premiers tours pour les trois valeurs de ω. À l'inverse, les trois courbes rejoignent en fin de partie celle de la méthode MCTS sans stratégies, indiquant l'application de la stratégie globale pour les dernières actions.

D'autre part, les résultats dans la table 6.2 indiquent que les meilleures performances du programme sont atteintes lorsque le coefficient d'exploration stratégique est égal à 0 : c'est à dire lorsqu'il n'y a pas d'exploration. Il est possible d'interpréter ce constat de deux manières. Tout d'abord rien n'impose que plusieurs objectifs soient progressivement avancés en parallèle, dans un jeu solitaire. Au contraire, il est même préférable de vider une première zone avant de passer à la suivante. Ensuite les simulations utilisées, lors de l'exploration pour confirmer les valeurs correspondantes à une stratégie, représentent autant de simulations perdues pour approfondir une stratégie locale en particulier. C'est d'ailleurs pour cette même raison, que nous avions choisi de mutualiser les évaluations globales dans le jeu de Go.

6.3.4.3. Discussion

Les expériences présentées ont été menées afin d'étudier l'influence des stratégies locales sur MCTS. Pour chaque jeu, le modèle applicatif était relativement simple afin de mettre en avant le modèle commun sur les spécificités de chacune des applications. Les résultats observés montrent que le modèle proposé répond aux attentes évoquées bien que les performances globales du programme restent encore modestes. Nous revenons dans cette section sur plusieurs limitations de la méthode actuelle, mises en lumière par les résultats obtenus et pouvant notamment expliquer les performances atteintes. À chaque fois, nous évoquons des améliorations généralement en lien avec les extensions envisagées en section 6.2.3.

Dépendant du modèle applicatif Cette méthode pose les bases d'une dynamique d'apprentissage adaptative pour MCTS autour d'objectifs intermédiaires. Cependant les objectifs intermédiaires et leur dynamique d'apprentissage associée (rétroaction, politique de *descent*) restent encore à établir par le concepteur *a priori*. Les performances obtenues sont étroitement liées à la pertinence des objectifs intermédiaires retenus. Par exemple pour le jeu de Go, nous avons ici proposé des stratégies très simples pour un plateau 9x9 qui ne seraient pas appropriées pour traiter les configurations complexes apparaissant dans un plateau 19x19. Les extensions proposées dans la section 6.2.3.3 permettraient, dans un premier temps, d'adapter les objectifs intermédiaires à la situation. À plus long terme, il serait souhaitable que les objectifs intermédiaires soient identifiés par l'agent lui-même.

Impact des évaluations locales Les évaluations locales permettent d'orienter la recherche autour d'objectifs intermédiaires mais cette influence reste encore trop

faible. Pour un coefficient stratégique ω égal à 1, nous avons vu dans la figure 6.12 que la méthode percevait la condition de terminaison seulement après avoir appliqué cette même stratégie pendant 7 tours consécutifs. À cet égard plusieurs améliorations sont envisageables. Tout d'abord, il est possible d'adapter la politique de *growth* ou de *roll-out* suivant les stratégies locales afin de privilégier certaines actions en rapport avec l'objectif intermédiaire. Ensuite le calcul de la rétroaction spécifique à la stratégie locale, est systématiquement perçu en fin d'épisode et tient compte du bruit introduit par les simulations aléatoires : pour le jeu CSI toute simulation réussira systématiquement à vider au moins la moitié d'une zone. Suivant les objectifs intermédiaires considérés, il serait parfois souhaitable de calculer cette valeur plus tôt au cours de l'épisode afin de privilégier l'impact des actions au cours de la phase *descent*, contrairement à celles réalisées dans la phase de *roll-out*.

Vers un arbre des stratégies locales Dans le modèle proposé, nous avons choisi de n'appliquer qu'une seule stratégie locale par épisode dans un premier temps. Les résultats obtenus pour le jeu du CSI mettent en lumière les limites d'une telle restriction. En effet nous avons vu, au travers de la profondeur maximale de l'arbre, que l'application de la stratégie globale dans les premiers tours revient à disperser la recherche pour ce jeu. Ainsi dès que l'agent a identifié une séquence d'actions permettant d'atteindre l'objectif, la profondeur de la recherche décroît car la stratégie globale ensuite appliquée n'arrive pas à maintenir l'avance gagnée. Les extensions présentées dans la section 6.2.3.2 constituent une perspective prometteuse pour un jeu solitaire comme le CSI.

Important volume de simulations requis Le modèle avec stratégie locale présenté en section 6.2 requiert un grand volume de simulations pour bénéficier à MCTS. Tout d'abord chaque stratégie dispose de sa propre arborescence indépendante. Par conséquent des actions pertinentes indépendamment de toute stratégie doivent être systématiquement redécouvertes dans chacune des arborescences. Cette première difficulté nous a notamment amené à mutualiser les évaluations globales dans l'application au jeu de Go au détriment d'un biais. Les extensions présentées en section 6.2.3.1 pour mutualiser la connaissance, permettraient de pallier ces limitations. Ensuite les récompenses perçues à la fin de chaque épisode répondent à deux attentes différentes mais dépendantes. Tout d'abord elles servent à améliorer la résolution de l'objectif poursuivi par le renforcement des états de son arborescence. Dans le même temps, cette récompense sert à évaluer la stratégie dans son ensemble pour la phase *strategic*. Cependant, pour correctement évaluer une stratégie, il faut pouvoir lui allouer un nombre suffisant de simulations afin qu'elle fasse ses preuves. Cette ambiguïté dans la récompense perçue rend particulièrement difficile le compromis entre exploration et exploitation au niveau de la phase *strategic*. À cet égard, une amélioration possible serait de tenir compte, dans les évaluations, des stratégies

d'autres récompenses complémentaires, comme par exemple la profondeur atteinte par l'arborescence.

6.4 Synthèse

Au cours de ce chapitre, nous avons étendu la dynamique d'apprentissage de MCTS en une méta-dynamique d'apprentissage. Nous avons proposé un modèle permettant d'orienter dynamiquement la recherche autour d'un panel d'objectifs intermédiaires. Comme nous l'avons évoqué au cours des chapitres précédents, la dynamique d'apprentissage de MCTS poursuit un seul et même objectif final. Cependant dans des situations comportant plusieurs sous-problèmes difficiles, la recherche se disperse entre les deux problèmes sans réussir à les traiter correctement. Nous pensons qu'une perspective prometteuse consiste à réifier la poursuite d'objectifs intermédiaires dans MCTS. En effet chaque objectif intermédiaire permet de concentrer momentanément la recherche afin de traiter correctement un sous-problème difficile, par une stratégie spécifique. Dans le même temps, l'agent sélectionne quelle stratégie appliquer suivant l'évolution de la partie, au travers d'une dynamique d'apprentissage plus générale.

Le modèle présenté a été mis en application pour deux jeux différents : les jeux du Go et du Clobber Solitaire Impartial. Malgré des performances globales encore modestes, les résultats obtenus montrent que MCTS avec des stratégies locales applique avec succès plusieurs stratégies successives lui permettant d'approfondir sa recherche localement. Ces résultats encourageants attestent de la capacité du modèle à réifier ces objectifs intermédiaires et nous invitent à l'approfondir afin d'en améliorer les performances globales. Le modèle présenté suggère de nombreuses pistes d'amélioration. En effet les représentations associées à la résolution de chaque objectif intermédiaire sont encore indépendantes et induisent un important surcoût d'expériences requises. Par ailleurs les stratégies spécifiques sont introduites *a minima* et le modèle ne permet pas de les combiner à l'image d'une séquence d'action. L'application de ce modèle à d'autres méthodes inspirées de MCTS comme NRPA permettraient de les considérer justement sur l'ensemble de la partie. Enfin en l'état, les objectifs intermédiaires sont laissés à la discrétion du concepteur faisant ainsi reposer le succès de la méthode sur la pertinence des objectifs retenus. Une perspective à plus long terme serait que l'agent soit en mesure d'adapter ses objectifs intermédiaires, voire de les élaborer au cours de l'apprentissage.

Nous avons choisi d'appliquer ce modèle aux jeux de Go et du Clobber Solitaire Impartial en particulier, afin d'en analyser le comportement. Néanmoins comme BHRF, ce modèle général est applicable à n'importe quel autre jeu. Les objectifs intermédiaires doivent être cependant conçus au cas par cas. Pour le jeu de Go en

particulier, les programmes avant l'avènement de MCTS reposaient sur une organisation semblable de la recherche autour d'objectifs intermédiaires. Les méthodes alors développées pourraient être ré-intégrées à MCTS au travers de ce modèle.

Conclusion

Au cours de cette thèse, nous nous sommes intéressé à la méthode Monte Carlo Tree Search (MCTS). Cette méthode relativement récente a permis des améliorations significatives dans le domaine des jeux combinatoires, grâce à une exploration performante de l'espace de recherche. La recherche a évolué bien plus vers une optimisation de l'efficacité du parcours que dans le sens d'une compréhension des mécanismes de cognition. Une telle compréhension apporterait cependant un appui précieux pour une fouille efficace. À titre d'exemple, les meilleurs joueurs humains qui ne disposent pas des mêmes capacités de calcul des ordinateurs, parviennent néanmoins à de meilleures performances dans des jeux difficiles comme le Go.

À ce jour, l'amélioration de cette méthode de parcours suivant une perspective purement combinatoire apparaît comme un problème particulièrement difficile. C'est la raison pour laquelle ce travail propose d'apporter une interprétation transversale de MCTS et de ses améliorations, selon une perspective d'inspiration cognitiviste. Nous avons approfondi les notions d'abstraction des représentations ainsi que les diverses rétroactions participant à l'apprentissage. En particulier, la notion de dynamique d'apprentissage nous a servi tout au long de ce manuscrit à mettre en avant les différentes interactions entre les représentations et les processus de résolutions internes à l'agent. Ce travail se veut ainsi original mais également exploratoire, en traitant grâce à deux propositions la problématique de l'accumulation de connaissances et de la réification de stratégies au sein de MCTS. Ces deux contributions s'illustrent notamment au travers des modifications apportées à la dynamique d'apprentissage de la méthode standard.

La première contribution propose de valoriser les connaissances déjà apprises par la méthode, en venant renforcer la dynamique d'apprentissage actuelle. Cette meilleure assimilation des connaissances permet notamment de capitaliser l'apprentissage sur plusieurs tours tout en contribuant à améliorer la qualité des simulations produites. Par cette méthode, les performances d'un programme de Go de niveau professionnel ont été améliorées de 11%. Ces résultats démontrent une meilleure capacité à exploiter les connaissances accumulées. Nous avons montré que grâce à ces connaissances, le système est plus autonome : il retrouve de lui-même des heuristiques généralement

codées par des experts. Nous avons comparé les performances de notre proposition avec ou sans heuristiques expertes complémentaires, et les résultats soulignent sans équivoque l'efficacité de notre proposition lorsque les heuristiques ne sont pas disponibles. En souhaitant développer l'autonomie cognitive du système, nous avons également dû considérer et reporter des temps de calcul associés supérieurs. À cet effet, nous proposons plusieurs optimisations afin d'améliorer l'efficacité de la méthode, aussi bien au niveau du modèle que de la représentation en mémoire. Il est également à noter que la parallélisation de masse aujourd'hui appliquée à MCTS, fera probablement évoluer les indicateurs de performance de ce type d'approche dans un avenir proche. On peut envisager que le doublement du temps de calcul pourrait à terme être absorbé par les capacités de parallélisation. L'enjeu porterait alors sur la qualité des connaissances accumulées plutôt que sur leur quantité.

La deuxième contribution propose d'adapter à la volée la dynamique d'apprentissage autour d'objectifs intermédiaires définis au préalable. Notre proposition exploite les capacités de fouille de MCTS tout en permettant de structurer sa recherche. La réification d'objectifs intermédiaires permet notamment à l'agent de concentrer momentanément sa recherche autour de problèmes relativement simples pris indépendamment mais devenant complexes une fois intégrés dans le contexte général de la partie. Cette proposition soulève de nouvelles questions : quelle rétroaction utiliser ? Comment considérer plusieurs objectifs intermédiaires consécutifs ? Peut-on parler de méta-recherche ? Nous avons pu constater pour le jeu du Clobber Solitaire Impartial, que le modèle proposé réussissait avec succès à concentrer la recherche à la volée au cours de la partie suivant les objectifs intermédiaires fournis. Ces résultats encourageants constituent un premier pas vers le développement de méthodes d'apprentissage encore plus adaptatives, et ouvrent ainsi la voie à de multiples extensions.

Bibliographie

[All94] Louis Victor ALLIS. « Searching for Solutions in Games and Artificial Intelligence ». Thèse de doct. Maastricht, Netherland : Limburg University, 1994 (cité en pages 4, 17).

[Bai10] Hendrik BAIER. « Adaptive Playout Policies for Monte-Carlo Go ». Mém.de mast. Institut für Kognitionswissenschaft, Universität Osnabrück, 2010 (cité en page 135).

[Bas+14] J. BASALDÚA, S. STEWART, J. M. MORENO-VEGA et P. DRAKE. « Two Online Learning Playout Policies in Monte Carlo Go : An Application of Win/Loss States ». In : *IEEE Transactions on Computational Intelligence and AI in Games* 6.1 (2014), p. 1–9 (cité en pages 52, 56, 78, 79, 89, 135–137).

[Bau11] Peter BAUDI. « MCTS with Information Sharing ». Mém.de mast. Charles University in Prague - Faculty of Mathematics et Physics, 2011 (cité en page 135).

[BC01] Bruno BOUZY et Tristan CAZENAVE. « Computer Go : An AI oriented survey ». In : *Artificial Intelligence* 132.1 (2001), p. 39–103 (cité en pages 7, 23, 39, 45, 147).

[BC12] Sébastien BUBECK et Nicolò CESA-BIANCHI. « Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems ». In : *Foundations and Trends® in Machine Learning* 5.1 (2012), p. 1–122 (cité en page 190).

[BD10] H. BAIER et P. DRAKE. « The Power of Forgetting : Improving the Last-Good-Reply Policy in Monte-Carlo go ». In : *IEEE Transactions on Computational Intelligence and AI in Games* 2.4 (2010), p. 303–309 (cité en pages 52, 56, 77, 119, 133).

[BDG13] Laurent BEAUDOU, Eric DUCHÊNE et Sylvain GRAVIER. « A survey about Solitaire Clobber ». In : *Games of No Chance* 4 (2013) (cité en pages 18, 164, 189).

[BG12] P. BAUDIŠ et J.L. GAILLY. « Pachi : State of the art open source Go program ». In : *Advances in Computer Games*. Springer-Verlag, 2012, p. 24–38 (cité en pages 69, 72).

[BH04] Bruno BOUZY et Bernard HELMSTETTER. « Monte Carlo go developments ». In : *Advances in computer games* 10 (2004), p. 159–174 (cité en page 11).

[Bou+10] A. BOURKI, G. CHASLOT, M. COULM et al. « Scalability and Parallelization of Monte-Carlo Tree Search ». In : *Proceedings of Advance in Computer Games 13*. 2010 (cité en page 12).

[Bou05] Bruno BOUZY. « Associating domain-dependent knowledge and Monte Carlo approaches within a Go program ». In : *Information Sciences* 175.4 (2005). Heuristic Search and Computer Game Playing {IV}, p. 247–257 (cité en pages 11, 44, 46, 56).

[Bro+12] C. BROWNE, E. POWLEY, D. WHITEHOUSE et al. « A Survey of Monte Carlo Tree Search Methods ». In : *Computational Intelligence and AI in Games, IEEE Transactions on* 4.1 (2012), p. 1–43 (cité en pages 51, 56, 66).

[Brü93] B. BRÜGMANN. « Monte carlo go ». In : *White paper* (1993) (cité en page 11).

[Caz09] T. CAZENAVE. « Nested Monte-Carlo Search. » In : *IJCAI*. T. 9. 2009, p. 456–461 (cité en pages 51, 56, 62).

[Caz12] Tristan CAZENAVE. « Monte Carlo Beam Search ». In : *Computational Intelligence and AI in Games, IEEE Transactions on* 4.1 (2012), p. 68–72 (cité en page 62).

[CBK08] Benjamin E CHILDS, James H BRODEUR et Levente KOCSIS. « Transpositions and move groups in Monte Carlo tree search ». In : *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE. 2008, p. 389–395 (cité en page 70).

[CH05] T. CAZENAVE et B. HELMSTETTER. « Combining tactical search and Monte-Carlo in the game of Go ». In : *IEEE CIG* (2005), p. 171–175 (cité en page 147).

[Cha+08] G.M.J-B CHASLOT, M.H.M WINANDS, H.J. VAN DEN HERIK, J.W.H.M UITERWIJK et B. BOUZY. « Progressive Strategies for Monte-Carlo Tree Search ». In : *New Mathematics and Natural Computation* 4.03 (2008), p. 343–357 (cité en page 71).

[Com09] Comunnauté du COMPUTER-GO. *Computer-Go mailing-list*. Oct. 2009 (cité en pages 6, 39).

[Cou07a] R. COULOM. « Computing Elo Ratings of Move Patterns in the Game of Go ». In : *Computer Games Workshop, Amsterdam, The Netherlands*. 2007 (cité en pages 39, 40, 46, 48, 56, 71, 72).

[Cou07b] R. COULOM. « Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search ». In : *Computers and Games* (2007), p. 72–83 (cité en page 59).

[Cou09] Rémi COULOM. *Criticality : a Monte-Carlo Heuristic for Go Programs*. Invited talk at the University of Electro-Communications, Tokyo, Japan. 2009 (cité en pages 68, 69).

[CS14] Christopher CLARK et Amos J. STORKEY. « Teaching Deep Convolutional Neural Networks to Play Go ». In : *CoRR* abs/1412.3409 (2014) (cité en pages 41, 42, 48, 56).

[DDF04] Erik D. DEMAINE, Martin L. DEMAINE et Rudolf FLEISCHER. « Solitaire Clobber ». In : *Theoretical Computer Science* 313.3 (2004). Algorithmic Combinatorial Game Theory, p. 325–338 (cité en page 189).

[Dem+02] Erik D. DEMAINE, Rudolf FLEISCHER, Aviezri FRAENKEL et Richard NOWAKOWSKI. *Dagstuhl Seminar on Algorithmic Combinatorial Game Theory.* Rapp. tech. Seminar no. 02081, report no 334, mar. 2002 (cité en page 18).

[DGM09] Eric DUCHÊNE, Sylvain GRAVIER et Julien MONCEL. « New results about impartial solitaire clobber ». In : *RAIRO - Operations Research* 43 (04 oct. 2009), p. 463–482 (cité en page 189).

[DN13] Madalina M DRUGAN et Ann NOWE. « Designing multi-objective multi-armed bandits algorithms : A study ». In : *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE. 2013, p. 1–8 (cité en page 155).

[Dra09] P. DRAKE. « The Last-Good-Reply Policy for Monte-Carlo Go ». In : *International Computer Games Association Journal* 32.4 (2009), p. 221–227 (cité en pages 36, 77, 82, 119).

[DU07] Peter DRAKE et Steve UURTAMO. « Heuristics in Monte Carlo Go. » In : *IC-AI*. Citeseer. 2007, p. 171–175 (cité en page 76).

[Enz+09] M. ENZENBERGER, M. MULLER, B. ARNESON et R. SEGAL. « FUEGO - An Open-Source Framework for Board Games and Go Engine Based on Monte-Carlo Tree Search ». In : *IEEE Transactions on Computational Intelligence and AI in Games* 2.4 (2009), p. 259–270 (cité en pages 39, 45, 56, 71, 72, 81, 123, 166).

[Enz03] Markus ENZENBERGER. « Evaluation in Go by a Neural Network using Soft Segmentation ». In : *Advances in Computer Games : Many Games, Many Challenges* (2003), p. 97–108 (cité en pages 42, 43, 51, 56).

[FB10] Hilmar FINNSSON et Yngvi BJÖRNSSON. « Learning Simulation Control in General Game-Playing Agents. » In : *AAAI*. T. 10. 2010, p. 954–959 (cité en pages 76, 77).

[GDD08] Sylvain GRAVIER, Paul DORBEC et Eric DUCHÊNE. « Solitaire Clobber Game on Hamming graphs. » In : *INTEGERS Elect. J. on Combin. Number Th.* 8.GO3 (2008), 21 pages (cité en page 189).

[Gel+06] S. GELLY, Y. WANG, R. MUNOS et O. TEYTAUD. *Modification of UCT with Patterns in Monte-Carlo Go*. Rapp. tech. INRIA, 2006 (cité en pages 38, 46, 72).

[Gel+12] S. GELLY, L. KOCSIS, M. SCHOENAUER et al. « The Grand Challenge of Computer Go : Monte Carlo Tree Search and Extensions ». In : *Communications of the ACM* 55.3 (2012), p. 106–113 (cité en pages 11, 12).

[Gel07] Sylvain GELLY. « Une contribution à l'apprentissage par renforcement ; application au Computer-Go. » Thèse de doct. Université Paris Sud, 2007 (cité en page 1).

[GP14] Tobias GRAF et Marco PLATZNER. « Common fate graph patterns in Monte Carlo Tree Search for computer go ». In : *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. Août 2014, p. 1–8 (cité en pages 40, 48, 56).

[Gra+01] T. GRAEPEL, M. GOUTRIE, M. KRÜGER et R. HERBRICH. « Learning on Graphs in the Game of Go ». In : *Artificial Neural Networks—ICANN 2001* (2001), p. 347–352 (cité en page 31).

[GS07] S. GELLY et D. SILVER. « Combining Online and Offline Knowledge in UCT ». In : *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, p. 273–280 (cité en pages 52, 74).

[GS08] S. GELLY et D. SILVER. « Achieving master level play in 9× 9 computer go ». In : *Proceedings of AAAI*. 2008, p. 1537–1540 (cité en pages 12, 71).

[GS11] S. GELLY et D. SILVER. « Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go ». In : *Artificial Intelligence* (2011) (cité en pages 68, 73).

[GSP14] Tobias GRAF, Lars SCHAEFERS et Marco PLATZNER. « On Semeai Detection in Monte-Carlo Go ». In : *Computers and Games*. Springer, 2014, p. 14–25 (cité en page 159).

[HBS12] Michael HARRÉ, Terry BOSSOMAIER et Allan SNYDER. « The Perceptual Cues that Reshape Expert Reasoning ». In : *Scientific Reports* 2 (11 juil. 2012) (cité en page 6).

[HM13] A. HUANG et M. MÜLLER. « Investigating the Limits of Monte Carlo Tree Search Methods in Computer Go ». In : *The 8th international conference on Computers and Games (CG2013)*. 2013 (cité en pages 83, 145).

[HP09] D.P. HELMBOLD et A. PARKER-WOOD. « All-Moves-As-First Heuristics in Monte-Carlo Go ». In : *Proceedings International Conference Artificial Intelligence, Las Vegas, Nevada*. 2009, p. 605–610 (cité en page 68).

[KCL98] Mark G. KARPOVSKY, Krishnendu CHAKRABARTY et Lev B. LEVITIN. « On a New Class of Codes for Identifying Vertices in Graphs ». In : *IEEE Transactions on Information Theory* 44.2 (1998), p. 599–611 (cité en page 33).

[KS06] L. KOCSIS et C. SZEPESVÁRI. « Bandit based Monte-Carlo Planning ». In : *Machine Learning : ECML 2006* (2006), p. 282–293 (cité en pages 67, 68).

[Lan08] Leonardo LANA DE CARVALHO. « Représentations émergentes : Une approche multi-agents des systèmes complexes adaptatifs en psychologie cognitive ». Thèse de doct. Université Lumière Lyon 2, 2008 (cité en pages 6, 8).

[Lec+98] Y. LECUN, L. BOTTOU, Y. BENGIO et P. HAFFNER. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (nov. 1998), p. 2278–2324 (cité en page 41).

[Lew11] Lukas LEW. « Modeling Go Game as a Large Decomposable Decision Process ». Thèse de doct. Warsaw University - Faculty of Mathematics, Informatics et Mechanics, 2011 (cité en pages 1, 12, 52, 77, 101, 137).

[Lit94] Michael L. LITTMAN. « Markov Games as a Framework for Multi-Agent Reinforcement Learning ». In : *In Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994, p. 157–163 (cité en page 51).

[Lor08] Richard J LORENTZ. « Amazons discover Monte-Carlo ». In : *Computers and games*. Springer, 2008, p. 13–24 (cité en page 70).

[Luc87] Édouard LUCAS. « Récréations Scientifiques ». In : *La Nature. Revue des sciences et de leurs applications aux arts et à l'industrie. Journal hebdomadaire illustré. Suivi de : Bulletin météorologique de La Nature, Boîte aux lettres, Nouvelles scientifiques*. Sous la dir. de Gaston TISSANDIER. T. 1887 : Quinzième année, deuxième semestre : n. 731 à 756. Masson, 1887, p. 402–404 (cité en page 34).

[Mad+14] Chris J. MADDISON, Aja HUANG, Ilya SUTSKEVER et David SILVER. « Move Evaluation in Go Using Deep Convolutional Neural Networks ». In : *CoRR* abs/1412.6564 (2014) (cité en pages 41, 43, 48, 56).

[MC08] Jean MEHAT et Tristan CAZENAVE. « Monte-Carlo Tree Search for General Game Playing ». In : *Université Paris 8 - LIASD* 8 (2008) (cité en page 11).

[Mül10] Martin MÜLLER. *Fuego-GB Prototype at the Human machine competition in Barcelona 2010 : a Tournament Report and Analysis*. Rapp. tech. Deptartement of Computing Science, University of Alberta, Canada, 2010 (cité en pages 29, 36, 83).

[Pel+09] S. PELLEGRINO, A. HUBBARD, J. GALBRAITH, P. DRAKE et Y.-P. CHEN. « Localizing Search in Monte-Carlo Go Using Statistical Covariance ». In : *ICGA Journal* 32 :3 (2009), p. 154–160 (cité en pages 68, 69).

[RN10] Stuart J. RUSSEL et Peter NORVIG. *Artificial intelligence : a modern approach*. Pearson education, 2010 (cité en page 34).

[Ros11a] Christopher D ROSIN. « Nested rollout policy adaptation for Monte Carlo tree search ». In : *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*. AAAI Press. 2011, p. 649–654 (cité en page 62).

[Ros11b] ChristopherD. ROSIN. « Multi-armed bandits with episode context ». English. In : *Annals of Mathematics and Artificial Intelligence* 61.3 (2011), p. 203–230 (cité en page 163).

[Rou14] Lisa ROUGETET. « Des récréations arithmétiques au corps des nombres surréels et à la victoire d'un programme aux échecs : une histoire de la théorie des jeux combinatoires au XXème siècle ». Thèse de doct. Université de Lille 1, 2014 (cité en page 1).

[RS11] R. RAMANUJAN et B. SELMAN. « Trade-Offs in Sampling-Based Adversarial Planning. » In : *ICAPS*. 2011, p. 202–209 (cité en pages 83, 118).

[RT10] A. RIMMEL et F. TEYTAUD. « Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search ». In : *Applications of Evolutionary Computation* (2010), p. 201–210 (cité en pages 77, 79).

[RTT11] A. RIMMEL, F. TEYTAUD et O. TEYTAUD. « Biasing Monte-Carlo Simulations through RAVE Values ». In : *Computers and Games* (2011), p. 59–68 (cité en page 75).

[Sai+07] Jahn-Takeshi SAITO, Mark HM WINANDS, JWHM UITERWIJK et H Jaap van den HERIK. « Grouping nodes for Monte-Carlo tree search ». In : *Computer Games Workshop*. 2007, p. 276–283 (cité en page 70).

[SB98] Richard S. SUTTON et Andrew G. BARTO. *Reinforcement Learning : An Introduction*. Cambridge Univ Press, 1998 (cité en pages 11, 24, 51, 54, 118, 119).

[Sch+07] Jonathan SCHAEFFER, Neil BURCH, Yngvi BJÖRNSSON et al. « Checkers Is Solved ». In : *Science* 317.5844 (2007), p. 1518–1522. eprint : `http://www.sciencemag.org/content/317/5844/1518.full.pdf` (cité en page 4).

[Sch+08] MaartenP.D. SCHADD, MarkH.M. WINANDS, H.Jaap van den HERIK, GuillaumeM.J.-B. CHASLOT et JosW.H.M. UITERWIJK. « Single-Player Monte-Carlo Tree Search ». English. In : *Computers and Games*. Sous la dir. de H.Jaap van den HERIK, Xinhe XU, Zongmin MA et MarkH.M. WINANDS. T. 5131. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, p. 1–12 (cité en page 166).

[Sch89] Jonathan SCHAEFFER. « The History Heuristic and Alpha-Beta Search Enhancements in Practice ». In : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 11.11 (1989), p. 1203–1212 (cité en pages 74–76).

[SDS94] Nicol N SCHRAUDOLPH, Peter DAYAN et Terrence J SEJNOWSKI. « Temporal Difference Learning of Position Evaluation in the Game of Go ». In : *Advances in Neural Information Processing Systems* (1994), p. 817–817 (cité en page 41).

[Seg11] Richard B SEGAL. « On the scalability of parallel UCT ». In : *Computers and Games*. Springer, 2011, p. 36–47 (cité en page 12).

[SHG06] D. STERN, R. HERBRICH et T. GRAEPEL. « Bayesian Pattern Ranking for Move Prediction in the Game of Go ». In : *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, p. 873–880 (cité en pages 46, 48, 56).

[Shi11] Huang SHIH-CHIEH. « New Heuristics for Monte Carlo Tree Search Applied to the Game of Go ». Thèse de doct. National Taiwan Normal University - Department of Computer Science et Information Engineering, 2011 (cité en pages 1, 40, 72).

[Sil09] David SILVER. « Reinforcement learning and simulation-based search in computer Go ». Thèse de doct. Alberta, Canada : University of Alberta, 2009 (cité en pages 1, 68).

[SPS99] Richard S. SUTTON, Doina PRECUP et Satinder SINGH. « Between {MDPs} and semi-MDPs : A framework for temporal abstraction in reinforcement learning ». In : *Artificial Intelligence* 112.1–2 (1999), p. 181–211 (cité en pages 148, 156, 157).

[SSM07] David SILVER, Richard SUTTON et Martin MÜLLER. « Reinforcement Learning of Local Shape in the Game of Go ». In : *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India : Morgan Kaufmann Publishers Inc., 2007, p. 1053–1058 (cité en pages 52, 56).

[SSM12] D. SILVER, R.S. SUTTON et M. MÜLLER. « Temporal-Difference search in computer Go ». In : *Machine Learning* (2012), p. 1–37 (cité en page 52).

[ST09] D. SILVER et G. TESAURO. « Monte-Carlo Simulation Balancing ». In : *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. 2009, p. 945–952 (cité en page 72).

[Tes95] Gerald TESAURO. « Temporal Difference Learning and TD-Gammon ». In : *Commun. ACM* 38.3 (mar. 1995), p. 58–68 (cité en pages 41, 43, 49, 51, 56).

[TM11] David TOM et Martin MÜLLER. « Computational Experiments with the RAVE Heuristic ». In : *Computers and Games*. Springer, 2011, p. 69–80 (cité en page 68).

[TWB12] M.J.W. TAK, M.H.M. WINANDS et Y. BJORNSSON. « N-Grams and the Last-Good-Reply Policy applied in General Game Playing ». In : *Computational Intelligence and AI in Games, IEEE Transactions on* 4.2 (2012), p. 73–83 (cité en pages 52, 56, 77, 79).

[VM12] Gabriel VAN EYCK et Martin MÜLLER. « Revisiting Move Groups in Monte-Carlo Tree Search ». In : *Advances in Computer Games*. Springer, 2012, p. 13–23 (cité en page 70).

[WG07] Y. WANG et S. GELLY. « Modifications of UCT and sequence-like simulations for Monte-Carlo Go ». In : *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*. 2007, p. 175–182 (cité en pages 71, 72, 89).

[WS12] Weijia WANG et Michele SEBAG. « Multi-objective Monte-Carlo Tree Search ». In : *Asian conference on machine learning*. T. 25. 2012, p. 507–522 (cité en page 155).

Articles de l'auteur

[Fab+12] André FABBRI, Frédéric ARMETTA, Eric DUCHÊNE et Salima HASSAS. « A new self-acquired knowledge process for Monte Carlo Tree Search ». en. In : *Computer Games Workshop, ECAI (European Conference on Artificial Intelligence)*. Août 2012, p. 13–24 (cité en page 87).

[Fab+14] André FABBRI, Frédéric ARMETTA, Eric DUCHÊNE et Salima HASSAS. « Knowledge complement for Monte Carlo Tree Search : an application to combinatorial games ». en. In : *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. Nov. 2014, p. 997–1003 (cité en page 87).

[Fab+15] André FABBRI, Frédéric ARMETTA, Eric DUCHÊNE et Salima HASSAS. « A Self-Acquiring Knowledge Process for MCTS ». In : *International Journal on Artificial Intelligence Tools* (**(À paraître)** 2015) (cité en page 87).

Annexes

A

A.1 Résultats théoriques pour le jeu du Clobber Solitaire Impartial

Dans cette annexe, nous évoquons les résultats théoriques existants pour le jeu du Clobber Solitaire Impartial présenté dans le Chapitre 2.

Le jeu de Clobber est parfois qualifié de jeu « d'annihilation ». La partie démarre d'une configuration remplie de pierres qui sont progressivement retirées à chaque action. Le déplacement d'une pierre s'accompagne systématiquement de la suppression d'une autre pierre. Ainsi, l'élaboration de la plus longue séquence de jeu possible équivaut à réduire au minimum le nombre de pierres présentes sur le plateau. L'étude théorique du Clobber Solitaire revient à s'intéresser au nombre de pierres minimum qu'il est possible de laisser suivant la configuration initiale : la k-réductibilité (k-*reducible* en anglais).

Les résultats théoriques pour le jeu Clobber Solitaire Impartial ont été établis pour différentes formes de plateau et différentes dispositions de pierres initiales. Ils fournissent des bornes supérieures à la réductibilité voire la valeur exacte. Dans le chapitre 6, nous nous sommes intéressés aux configurations indiquées en gris.

Plateau	Configuration initiale	k-réductibilité	Référence
Grille k, n	départ en damier $k \times n \equiv 0 \bmod 3$	2 réductible (exact)	[DDF04]
Grille k, n	départ en damier $k \times n \not\equiv 0 \bmod 3$	1 réductible (exact)	[DDF04]
Grille k, n	départ non-monochrome $k \geqslant n \geqslant 4$	2 réductible (max)	[BDG13]
Graphe de Hamming	départ non-monochrome	2 réductible (max)	[GDD08]
Graphe aléatoire $G(n, p)$	$p \in]0, 1[$ $n \longrightarrow \infty$	1 réductible	[DGM09]

Table A.1.– Réductibilités démontrées pour le jeu de Clobber Solitaire Impartial [BDG13]

A.2 Problème de bandit à n-bras

Dans cette annexe, nous présentons plus en détails le problème canonique du bandit à n-bras évoqué dans le Chapitre 3 qui sert notamment de fondement théorique pour l'amélioration UCT présentée dans le Chapitre 4.

Le problème du bandit à n-bras résume le compromis entre exploration et exploitation sous sa forme la plus simple. Le terme de « bandit manchot » est employé pour désigner une machine à sous de casino. Par extension, un bandit à n-bras revient à placer un joueur de casino face à n machine à sous. Il s'agit d'un problème canonique habituellement étudié pour établir des preuves théoriques [BC12].

Un agent a le choix entre n actions possibles. Chacune de ses actions lui rapporte un gain aléatoire dont la loi est inconnue de l'agent *a priori*. L'agent est confronté de façon répétée à ce même choix. Chaque action réalisée par l'agent lui fournira une récompense mais aussi une information sur la probabilité de gain associée à cette action. Pour une itération fixée, l'agent aura le choix entre l'exploitation d'une action qui lui a fourni le meilleur gain jusqu'à présent, ou l'exploration d'une action sous-optimale. Une action actuellement considérée comme sous-optimale peut s'avérer optimale sur un plus long terme car les informations accumulées jusqu'alors étaient insuffisantes pour le repérer. Ainsi, l'objectif de cet agent est d'identifier l'action optimale le plus rapidement possible et ensuite de s'y tenir.

Une stratégie consiste à calculer pour chaque action des valeurs complémentaires en plus du gain moyen obtenu. Ces valeurs sont généralement appelées des index de Gittins. Les *Upper Confidence Bound* UCB sont des index de Gittins calculés de la manière suivante :

$$Q_{\text{UCB}}(a) = \bar{x}_a + \sqrt{2\frac{\ln n}{n_a}} \quad ,$$

tel que

$\bar{x}_a$	: gain moyen de l'action a	n	: nombre d'itérations
n_a	: nombre de sélections de l'action a		

À mesure que le nombre d'itérations n croît, les actions qui n'auront pas été fréquemment sélectionnées, c'est à dire pour une valeur de n_a faible, seront privilégiées. Leur calcul a été adapté pour le cas des arbres de recherches comme nous le verrons par la suite.

A.3 Différences de couverture entre MCTS et BHRF

Dans cette annexe nous revenons sur les schémas temporels couverts par BHRF (cf. Chapitre 5). Nous mettons ici en avant les différences entre l' extension incrémentale de l'arbre de recherche de MCTS et celle rétrograde réalisée par BHRF

Contrairement à ce qu'il serait possible d'imaginer, BHRF ne couvre pas nécessairement tous les schémas temporels observables dans l'arbre de recherche. L'ordre suivant lequel la recherche arborescente explore les états, a une influence sur la forme résultante de la forêt rétrograde de schémas. Par exemple supposons les trois itérations suivantes de MCTS dans la figure A.1. Nous construisons en parallèle la structure résultante de la forêt rétrograde de schémas. À la fin du troisième épisode BHRF ne comporte pas le schéma temporel $AB \rightarrow C$.

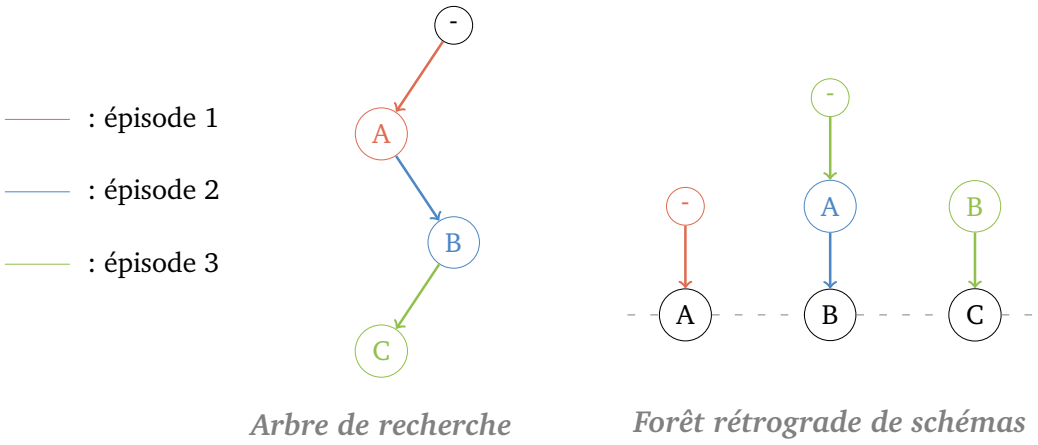


Figure A.1.– Exemple de construction de BHRF

Dans BHRF, l'accumulation d'un nouveau schéma requiert que tous les schémas plus petits soient déjà couverts pour cette même réponse, ailleurs dans la structure. L'extension de la structure est réalisée suivant un sens anti-chronologique, c'est à dire qu'un nouveau schéma est ajouté s'il comporte une action supplémentaire au préfixe d'un schéma déjà couvert. Dans l'arbre de recherche, un nouvel état peut être ajouté s'il est atteignable en une action depuis n'importe quel état déjà couvert. L'extension de la structure est réalisée suivant un sens chronologique, c'est à dire qu'un nouvel état est ajouté si la séquence pour l'atteindre comporte une action supplémentaire au suffixe d'une séquence déjà couverte.

Ainsi, lorsqu'un nouvel état couvert par l'arbre de recherche fait suite à une action qui n'a pas encore été exploré par ailleurs, la forêt de schéma rétrograde ne considérera le schéma temporel correspondant à cette séquence que lorsque tous les schémas temporels plus petits, et associés à cette réponse, auront été préalablement ajoutés.

Dans l'exemple présenté ci-dessus, le schéma -AB→C n'est pas ajouté car le schéma B→C n'était pas encore couvert par la forêt.

Les schémas temporels couverts par BHRF coïncideront avec tous ceux potentiellement couverts par l'arbre de recherche lorsque les mêmes sous-séquences d'actions se reproduisent fréquemment dans les différentes branches de l'arbre de recherche ; en d'autres termes lorsque la structure de l'arbre est symétrique (politique de *descent* exploratoire).

A.4 Mise à jour BHRF d'une forêt rétrograde de schémas

Dans cette annexe, nous précisons les algorithmes de BHRF (cf. Chapitre 5) pour renforcer une forêt rétrograde de schémas..

Nous regroupons ici les différents algorithmes nécessaires pour mettre à jour la forêt rétrograde de schémas à partir du même renforcement que celui fourni à l'arbre de recherche dans MCTS. Ces algorithmes correspondent aux différentes étapes de la procédure de renforcement décrite dans la section 5.2.2.1. Le premier algorithme 6 traite du renforcement de la forêt rétrograde de schémas, tandis que le second algorithme 7 traite du renforcement d'un unique arbre de cette même forêt.

Algorithm 6 Mise à jour (*update* et *growth*) - Forêt BHRF

```
procedure UPDATEREPLYFOREST(descentSequence : Array<Move>,
      outcome : float)
//descentSequence : moves selected in the last tree descent
//outcome : result of the simulation following the descent
    for i ← descentSequence.size −1 to 0 do
        rt : ReplyTree
        rt ← self.getReplyTree(descentSequence[i])
// the reply tree leading to move i is considered
        if opponentMove(i) then
            rt.updateTree(i,descentSequence,inv(outcome))
        else
            rt.updateTree(i,descentSequence,outcome)
        end if
    end for
end procedure
```

Algorithm 7 Mise à jour (*update* et *growth*) - Arbre de réponse

```
procedure UPDATETREE(i : int, descentSequence : Array<Move>,
    outcome : float)
//i : position of the reply move in the descentSequence
    nodeCreated : Boolean
    mv : Move
    nodeCreated ← false
    childNode, lastNode : Node
    lastNode ← self.getRoot()
// the root node of the tree corresponds to the reply
    while i > 0 && ¬ nodeCreated do
        i ← i −1
        mv ← descentSequence[i]
        childNode ← lastNode.getDirectChild(mv)
        if childMove == null then
            childMove ← lastNode.createChild(mv)
            childMove.updateMean(outcome)
            nodeCreated ← true
        else
            childMove.updateMean(outcome)
            lastNode ← childNode
        end if
    end while
end procedure
```

A.5 Politiques de *roll-out* avec BHRF

Dans cette annexe nous précisons les algorithmes des politiques de simulation de BHRF (cf. Chapitre 5)

Nous détaillons ici les algorithmes pour réutiliser les schémas temporels accumulés par BHRF lors de la phase *roll-out*. Ces algorithmes correspondent aux deux politiques différentes de BHRF pour des évaluations moyennes et volatiles respectivement. L'algorithme 8 formalise la politique de BHRF avec des évaluations moyennes. La déroulement de la procédure a été présentée dans la section 5.2.3. Cet algorithme repose sur une accumulation des schémas temporels dans une forêt rétrograde de schémas présentée dans la section 5.2.1.

Algorithm 8 Politique de *roll-out* pour BHRF

```
procedure ROLLOUTPOLICY(lastMoves : Array<Move>) : Move
//lastMoves : moves previously selected (temporal context)
    candidate : Move
    lstCandidate : List<Move>

//Probability ϵ to use BHRF
    if randomValue() < ϵ then
        for m : Move ∈ legalMoves() do
// Gets the best estimated candidate matching the context
            candidate = getReplyTree(m).BestCandidate(lastMoves)
            if candidate ≠ null then
                lstCandidate.add(candidate)
            end if
        end for
// Selects the reply according to Equation 5.1
        candidate = BestCandidateSoftMaxUct(lstCandidate)

        if randomValue() < candidate.uctEstimate() then
            return candidate
        end if
    end if
// Plays default policy otherwise
    return defaultRandomMove()
end procedure
```

L'algorithme 8 formalise la politique de BHRF avec des évaluations volatiles. Le déroulement de la procédure a été présenté dans la section5.4.3. Cet algorithme repose sur une accumulation de schémas temporels dans un arbre direct de schémas présenté dans la section 5.3.2. Dans la section 5.3.4.2 nous avons déjà présenté l'algorithme 5 formalisant la politique de BHRF avec des évaluations moyennes avec un arbre direct de schémas. Outre la sélection « volatile » des réponses, la légalité des réponses candidates n'a pas besoin d'être vérifiée au préalable, comme dans

l'algorithme 5. Nous nous contentons simplement de vérifier que le noeud atteint dans l'arbre direct de schémas dispose au moins d'une réponse.

Algorithm 9 Politique de *roll-out* incrémentale pour BHRF volatile

```
procedure ROLLOUTPOLICY(lastMove : Move, currentNode : Node ) : Move
//lastMove : move previously selected
//currentNode : current position in the BHRF forward tree
//(initialized according to the descent sequence)
    candidate : Move
    lstCandidate : List<Move>
// Progress through the tree
    currentNode ← progress(currentNode,lastMove)
//Probability ϵ to use BHRF
    if randomValue() < ϵ then
// Get legal childs
        for candidate : Node ∈ currentNode.Childs() do
            if candidate.isLegal() then
                lstCandidate.add(candidate)
            end if
        end for
// Get a context with at least a reply
        while !currentNode.hasChildren() do
            currentNode ← currentNode.BackTrackingNode()
        end while
// Get last updated reply
        candidate ← currentNode.getLastReply()
        if IsLegal(candidate) then
            return candidate
        end if
    end if
// Plays default policy otherwise
    return defaultRandomMove()
end procedure
```

A.6 Mise à jour BHRF avec des évaluations volatiles

Dans cette annexe nous précisons les différents algorithmes de BHRF (cf. Chapitre 5) pour renforcer un arbre direct de schémas (cf. section 5.4.2).

Nous regroupons ici les différents algorithmes nécessaires pour mettre à jour un arbre direct de schémas pour des évaluations volatiles. Ces algorithmes correspondent aux procédures de renforcements décrites dans la section 5.4.2. Le premier algorithme 10 traite du renforcement complet de l'arbre direct de schémas. La principale différence avec l'algorithme 1 présenté dans la section 5.3.3.2 p.107, réside dans l'ordre suivant lequel la progression dans l'arbre et le renforcement des noeuds sont réalisés.

La mise à jour BHRF comporte deux volets : le renforcement des schémas couverts par BHRF et l'accumulation de nouveaux schémas. Dans le cadre d'évaluations volatiles, seul le renforcement des schémas déjà couverts est modifié. Comme cela est présenté dans l'algorithme 11, nous supposons que chaque noeud de l'arbre direct de schémas est susceptible de mémoriser la dernière réponse renforcée.

L'algorithme 12 de création de noeud est rigoureusement le même que celui présenté dans la section 5.3.3.2.

Algorithm 10 Mise à jour de la séquence (*update*) - arbre direct de schémas

```
procedure UPDATESEQUENCEVOL(descentSequence : Array<Move>)
    currentNode,nextNode : Node
    currentNode ← self.getRoot()
//Follow descent sequence
    for i ← 0 to descentSequence.size do
//Create the node if it does not exist for the current one
        if nextNode == null then
            nextNode ← createNextNode(currentNode,descentSequence[i])
        end if
        updateBackTrackingNodesVol(currentNode,descentSequence[i])
        nextNode ← currentNode.getChild(descentSequence[i])
        currentNode ← nextNode
    end for
end procedure
```

Algorithm 11 Mise à jour d'un noeud volatile (*update*) - arbre direct de schémas

```
procedure UPDATEBACKTRACKINGNODESVOL(startNode : Node, nextMove : Move)
    currentNode,nextReply,backNode : Node
    currentNode ← startNode
    backNode ← currentNode.BackTrackingNode()
// Only the root node is its own backtracking node
    while currentNode != backNode do
// Update and backtrack
        nextReply ← currentNode.getChild(nextMove)
        currentNode.updateNextReply(nextReply)
        currentNode ← backNode
        backNode ← currentNode.BackTrackingNode()
    end while
end procedure
```

Algorithm 12 Création de noeud (*growth*) - arbre direct de schémas

```
procedure CREATENEXTNODE(fatherNode : Node, nextMove : Move) : Node
// fatherNode : node currently reached in the tree (Input and Output parameter)
    backNode : Node
// Check if the backtrackingnode of the new node exists
    backNode ← fatherNode.getBacktrackingNode()
    backNode ← backNode.getChild(Move)
    while backNode == null do
// Otherwise backtrack one more time
        fatherNode ← fatherNode.getBacktrackingNode()
        backNode ← fatherNode.getBacktrackingNode()
        backNode ← backNode.getChild(Move)
    end while
    return createNode(fatherNode,backNode, nextMove)
end procedure
```

Index