La Recherche Abstraite Graduelle de Preuve

Gradual Abstract Proof Search

T. Cazenave

Laboratoire d'Intelligence Artificielle

Dept Informatique, Université Paris 8, 2 rue de la liberté, 93526 St Denis Cedex
cazenave@ai.univ-paris8.fr

Résumé

Nous présentons un nouvel algorithme de développement d'arbres ET/OU dans les jeux. Il permet des gains très importants sur l'Alpha-Béta et sur les autres algorithmes existants. Il permet par exemple de résoudre le jeu d'AtariGo sur un damier 6x6 en moins de 10 minutes alors que l'Alpha-Béta demanderait des années. Les particularités de l'algorithme sont une très grande sélectivité dans le choix des coups à essayer, associée à une certitude de trouver le résultat correct. Il est basé sur une théorie des jeux combinatoires étendue aux valeurs inconnues. Il peut être utilisé pour résoudre complètement des jeux complexes comme Hex ou le Gomoku, mais aussi comme résolveur de problèmes à l'intérieur de programmes de jeux complexes. Il peut par exemple trouver efficacement des mats aux Echecs ou prouver la prise, la connexion ou la vie dans des situations complexes au jeu de Go. Nous présentons son application à AtariGo 6x6.

Mots Clef

Résolution de Jeux, Résolution de Problèmes, Jeu de Go.

Abstract

We present a new AND/OR tree search algorithm for games. It enables dramatic gains on Alpha-Beta and other existing algorithms. For example, it enables to solve the 6x6 AtariGo game in less than 10 minutes, where Alpha-Beta would spend years. The algorithm is very move selective, however it is assured to find the correct result. It is based upon a theory of combinatorial games extended to unknown values. It can be used to solve complex games such as Hex or Gomoku, and as a problem solver in complex game playing programs. It can prove mates in Chess, capture, connection and life in Go. We present its application to 6x6 AtariGo.

Keywords

Game Solving, Problem Solving, Game of Go.

1 Introduction

Nous présentons la recherche abstraite graduelle de preuve (AGPS pour Abstract Gradual Proof Search), un algorithme de résolution de jeux qui permet des gains très important par rapport à l'algorithme Alpha-Béta. Cet algorithme est une évolution de la recherche abstraite de preuve (APS pour Abstract Proof Search) [Cazenave 2000] et de l'algorithme d'élargissement itératif (IW pour Iterative Widening) [Cazenave 2001] qui permettent déjà des gains très substantiels sur l'algorithme Alpha-Béta. APS,IW et AGPS sont tous basés sur la notion de coups forcés [Cazenave 1996, 1998] : ce sont les coups que l'adversaire doit jouer aux nœuds Min de l'arborescence pour éviter de perdre rapidement, les coups forcés d'ordre N sont les coups qui empêchent d'atteindre le but en N coups. AGPS est un algorithme général qui peut s'appliquer à de nombreux jeux comme le jeu de Go, la recherche de mats au jeu d'Echecs, le jeu de Hex, le jeu de Gomoku, d'Abalone ainsi que de nombreux autres jeux. Il constitue un pas supplémentaire vers la conception d'algorithmes généraux et efficaces de recherche [Pitrat 1998]. Dans ce papier nous nous focalisons sur l'utilisation d'AGPS pour résoudre le jeu d'AtariGo sur damier 6x6. AtariGo est un jeu qui est utilisé pour initier les joueurs débutants au jeu de Go. Le but est de capturer le premier une pierre de l'adversaire. C'est un jeu qui comporte en moyenne plus de 20 coups légaux et dont la démonstration nécessite un arbre de profondeur 17. Il est donc hors d'atteinte des algorithmes classiques comme l'Alpha-Béta qui demanderait plusieurs années pour le résoudre, alors que AGPS le résout complètement en moins de 10 minutes, et permet par la suite d'engendrer un programme parfait qui répond quasi-instantanément à tous les coups.

La deuxième section explique le jeu d'AtariGo que nous avons pris comme exemple d'application. La troisième section montre comment il est possible de démontrer des théorèmes dans les jeux. La quatrième section traite de l'algorithme de recherche. La cinquième section présente les résultats expérimentaux.

2 AtariGo

AtariGo est un jeu qui est utilisé pour initier les joueurs débutants au jeu de Go. Le but est de capturer le premier une pierre de l'adversaire.

Il peut être joué sur un damier de taille arbitraire. En général, on choisit un petit damier pour que les parties ne soient pas trop longues. De même, on peut choisir de croiser deux pierres noires et deux pierres blanches au centre pour rendre la position plus instable. C'est cette dernière version que nous avons choisie de résoudre. On peut voir dans la figure 1 la position initiale sur un damier 6x6, qui est une des positions que nous avons complètement résolue.

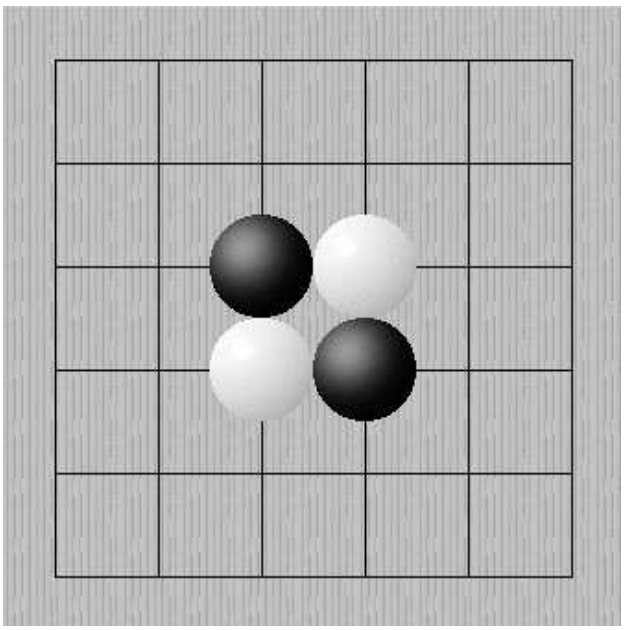

Figure 1. La position initiale d'AtariGo 6x6

Les règles sont similaires au jeu de Go, à savoir que Noir commence, que Noir et Blanc posent alternativement une pierre sur le damier, et que les ensembles de pierres de la même couleur qui sont reliées par les lignes du damier forment des chaînes de pierres. Le nombre d'intersections vides voisines d'une chaîne de pierres est son nombre de libertés. Une chaîne est capturée si elle n'a plus de libertés. Par exemple, dans la figure 1, les quatre chaînes de pierres ont toutes 2 libertés, alors que dans la figure 4, les quatre chaînes de pierres ont toutes 4 libertés. Une chaîne de pierres qui n'a plus qu'une liberté est en Atari , d'où le nom du jeu.

3 Jeux et Démonstration de Théorèmes

L'étude d'algorithmes généraux qui permettent de trouver des combinaisons gagnantes dans des jeux à deux joueurs, à information complète et pour des buts biens définis en sélectionnant un petit nombre de coups possibles remonte aussi loin que Genejeu [Pitrat 1969]. On y trouve déjà la notion d'ensemble de coups qui permettent de ne pas perdre. Notre algorithme de recherche est en partie basé sur ces idées que nous avons développées et affinées, donnant ainsi naissance à une famille d'algorithmes qui permettent de résoudre efficacement des jeux en exhibant une solution gagnante.

Une autre évolution importante par rapport aux algorithmes basés sur la manipulation d'un langage, comme dans Introspect [Cazenave 1996, 1998] ou dans Genejeu, est que nous avons décrit l'algorithme sous forme d'algorithme général de recherche arborescente ce qui le rend plus facilement implémentable qu'un générateur de programmes. Toutefois cet algorithme nécessite une analyse préalable du jeu auquel on l'applique pour trouver les ensembles complets de coups abstraits du jeu. Les ensembles de coups abstraits sont les ensembles complets de coups, décrits de manière abstraite, qui peuvent avoir une influence sur l'achèvement d'un but un nombre fixé de coups à l'avance.

Pour résoudre un jeu, un programme développe un arbre ET/OU. En général, le gain est représenté par un 1, la perte par un 0, et les états inconnus par 0,5 ce qui permet d'appliquer l'algorithme Alpha-Béta.

Les algorithmes que nous décrivons dans ce papier permettent de réduire considérablement la taille des arbres ET/OU nécessaires pour résoudre un jeu. Dans la première sous-section nous présentons notre notation pour les jeux combinatoires à valeurs inconnues. La deuxième sous-section définit les jeux graduels qui sont une extension des jeux combinatoires à valeurs inconnues. Dans la troisième sous-section nous expliquons comment on peut trouver les ensembles complets de coups abstraits, soit par une analyse purement abstraite, soit par une combinaison d'analyse abstraite et de récupération dynamique d'informations depuis les arbres de jeux.

3.1 Les Jeux Combinatoires à Valeurs Inconnues

La théorie des nombres de Conway ainsi que la théorie combinatoire des jeux se sont montrées très pertinentes pour de nombreux jeux [Conway & al., 1982]. Toutefois ces jeux restent souvent assez simples pour être complètement analysés. Lorsqu'on s'attaque à des jeux plus complexes qui ne peuvent être totalement analysés, on introduit alors la notion d'inconnue dans les jeux combinatoires [Cazenave 1996], ce qui permet par exemple de résoudre des jeux en n'explorant qu'une petite partie de l'arbre combinatoire total qui représente le nombre associé au jeu.

Dans l'extension de la théorie combinatoire des jeux aux valeurs inconnues, chaque feuille d'un arbre peut prendre trois valeurs : Gagné (g), Perdu (p) ou Inconnu (i). Ainsi une situation dans laquelle le joueur gauche (Noir par exemple) peut gagner s'il joue un coup, et qui devient inconnue si le joueur droit (Blanc) joue est représentée par l'arbre le plus à gauche de la figure 2. Nous associons le jeu gi à cette situation, qui est équivalent en notation de

Conway à { g | i }. Ce jeu est suivi de l'indice 1 parce que blanc atteint son but en un coup.
D'une manière générale le nombre de coups minimal contre la meilleure défense est donné en indice des jeux combinatoires à valeurs inconnues.

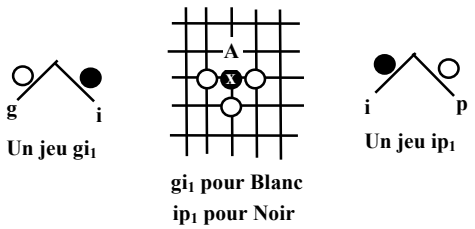

Figure 2. Jeux combinatoires à valeurs inconnues

On peut les définir de façon plus formelle. Soit P une position, c un coup et joue(P,m) la fonction qui renvoie la nouvelle position après le coup c sur la position P. La lettre A désigne le joueur Ami, et la lettre E le joueur Ennemi. On peut définir les jeux comme suit :

gi_k (P,A) = Ami peut gagner en k coups si Ami joue en premier dans la position P, et si les deux joueurs jouent parfaitement chacun leur tour. Autrement dit : ∃ coup Ami tel que { P_1=joue(P,coup), $g_{k-1}(P_1,A)$ }

ip_k (P,E) = gi_k (P,A).

S_k (P,E) est l'ensemble de tous les coups ennemis qui empêchent Ami de gagner en k coups ou moins dans la position P quand on a gi_k (P,A). Autrement dit: { coup Ennemis c sur P / { gi_k (P,A), { P_1=joue(P,c), ∀ o, o<=k, non ($gi_o(P_1,A)$) }}}.

g_k (P,A) = Ami peut gagner en k coups dans la position P même si Ennemi joue en premier, et si les deux joueurs jouent parfaitement chacun leur tour. Autrement dit : ∃ m { m<=k, ip_m (P,E), ∀ coup c ∈ S_m(P,E) { P_1=joue(P,c), ∃ o {o<=k, $gi_o(P_1,A)$ }}}.

Il a déjà été établi qu'il est possible d'engendrer automatiquement des programmes qui vérifient les définitions de ces jeux combinatoires en utilisant les règles du jeu définies dans un langage logique et un système de métaprogrammation [Cazenave, 1996,1998]. Les programmes engendrés donnent exactement les mêmes résultats que les fonctions basées sur un mécanisme de recherche. Ils peuvent être engendrés automatiquement en généralisant de façon fiable des traces de preuves sur des exemples auto-générés [Cazenave, 1996], ou statiquement en spécialisant les définitions des jeux combinatoires à valeurs inconnues sur les règles du jeu [Cazenave, 1998].

3.2 Les Jeux Graduels

Dans cette sous-section, nous présentons et nous nommons des jeux combinatoires graduels. Ils permettent de faire une distinction plus fine entre les jeux que les définitions des jeux gi_k, ip_k et g_k définis dans la sous-section précédente.

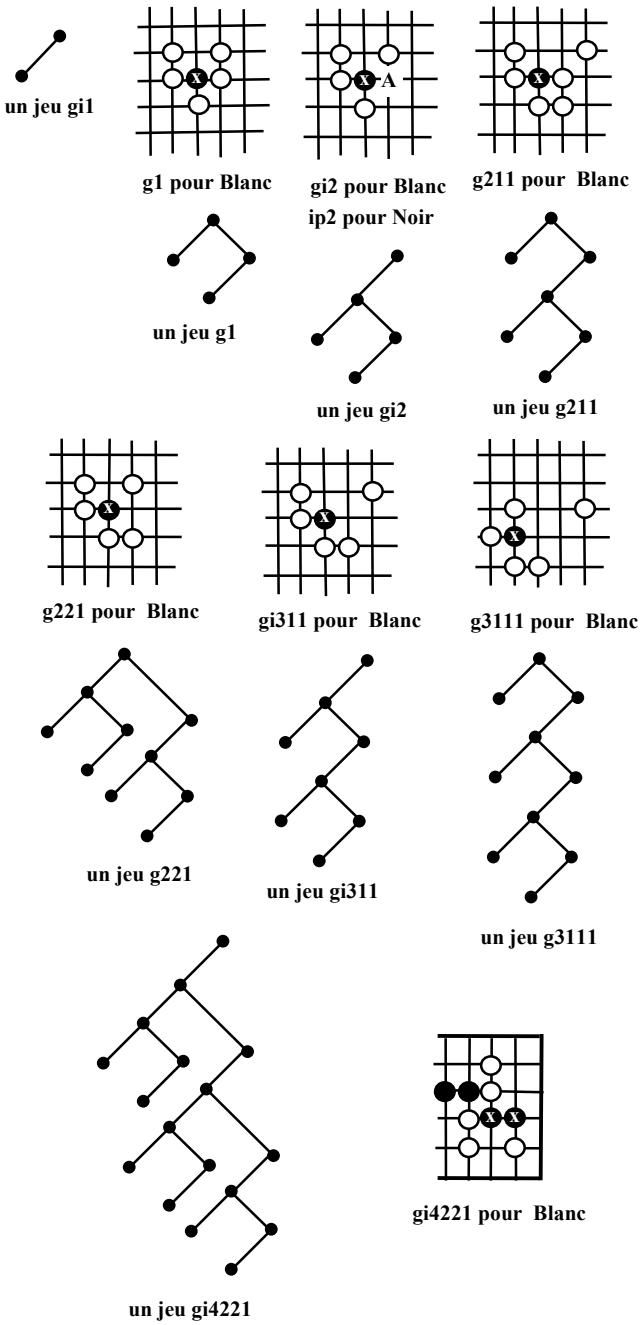

Figure 3. Exemples de Jeux Graduels

Un jeu graduel permet une représentation plus fine de l'état d'un jeu qu'un jeu combinatoire à valeurs inconnues

simple. Dans la section précédente, les jeux combinatoires à valeurs inconnues étaient associés à un nombre représentant le nombre minimal de coups du joueur qui veut atteindre le but. Or deux jeux ayant le même nombre minimal de coups peuvent avoir une complexité moyenne de vérification différente. Ainsi le jeu gi_{311} de la figure 3 contient 8 nœuds alors que le jeu gi_{321} (qui est le jeu qui amène à un jeu g_{221}) a 11 noeuds.

Pour présenter les jeux graduels, nous avons simplifié leur représentation graphique. Le joueur Gauche est toujours celui qui peut atteindre le but et les coups gagnants sont donc représentés par des branches qui vont vers la gauche à partir du nœud au-dessus. Les branches qui amènent à des états inconnus sont omises pour alléger la présentation. Ainsi le jeu gi_1 représenté par deux branches dans la figure 2 se transforme en un simple arc qui descend vers la gauche dans la figure 3. A coté du jeu gi_1, nous avons le jeu g_1 qui signifie que Gauche peut atteindre son objectif en un coup, quels que soient les coups de Droite, même si Droite joue en premier. Un diagramme de la figure 3 donne un exemple d'un pierre noire prise avec un jeu g_1 pour Blanc. La forme graphique du jeu g_1 est donnée en dessous. Elle signifie simplement que Gauche peut gagner s'il joue (arc de la racine vers la gauche), et que quel que soit le coup de Droite (arc de la racine vers la droite), Gauche a une réponse qui lui permet de gagner (arc vers la gauche suivant l'arc précédent).

On peut définir une taxonomie des jeux graduels. Elle permet de connaître les dépendances entre jeux graduels et ainsi d'éviter des recalculs inutiles. Pour définir la taxonomie, on définit la relation 'est un' entre deux jeux : un jeu A est un jeu B si en faisant coïncider les deux racines de la définition graphique des deux jeux, la représentation de B recouvre complètement celle de A, et aucun des nœuds intérieurs de A sans fils droit n'a de fils droit dans B. Ainsi les jeux gi_1 sont des jeux gi_2. Si un jeu est inclus un autre jeu, par exemple gi_1 est inclus dans gi_2, on sait qu'il n'est pas nécessaire de vérfier gi_2 si gi_1 est vérifié. De même, il est inutile de vérifier gi_1 si l'on sait qu'on va vérifier gi_2, puisque la vérification de gi_2 inclus la vérification de gi_1. Un autre exemple moins évident est qu'il n'est pas nécessaire de vérifier les jeux gi_{321} si on vérifie les jeux gi_{4221}.

Les jeux ip dont nous nous servirons par la suite, sont les symétriques des jeux gi. Ils consistent à vérifier que le jeu est gi pour Gauche, puis à trouver l'ensemble de tous les coups de Droite qui amènent à une position où le jeu n'est plus gi pour Gauche. Cet ensemble peut éventuellement être vide, dans ce cas on a montré que le jeu est g.

Les jeux graduels ont un grain plus fin que les jeux définis dans la section précédente. Ainsi les jeux ip_3 correspondent aux jeux graduels ip_{311} et ip_{321} réunis. Les jeux graduels permettent de trouver efficacement des coups forcés dans des situations où les jeux de la section précédente demanderaient beaucoup de temps. Par exemple, il est fréquent dans de nombreux jeux (AtariGo, Go, Hex, Phutball…) de rencontrer des jeux graduels ip_{51111}, ils sont détectés très rapidement par une classification graduelle, mais amènent à une grande perte de temps dans les algorithmes de recherche si on utilise seulement les jeux ip_5 pour les détecter, car les jeux ip_5 prennent beaucoup de temps pour être testés dans de nombreuses positions, alors que les jeux ip_{51111} sont toujours très rapides.

3.3 Ensembles complets de coups

Lorsqu'on essaie de trouver les coups associés à un jeu ip, on peut essayer tous les coups possibles. Toutefois il est beaucoup plus efficace dans certaines situations de sélectionner de façon sûre un ensemble restreint de coups. Par exemple, dans le cas des jeux ip_1, on a vérifié que la situation est gi_1, et donc qu'une chaîne de pierres n'a plus qu'une seule liberté et peut être prise au prochain coup. Les seuls coups intéressants à tester pour le jeu ip_1 sont les libertés des chaînes qui n'ont qu'une seule liberté. Il est inutile de regarder les autres coups car on sait à l'avance qu'ils ne marchent pas (qu'ils n'empêchent pas Gauche de gagner). De même on peut par analyse des règles du jeu trouver l'ensemble complet des coups à essayer pour les jeux ip2 : il suffit de regarder les libertés des chaînes de deux libertés ou moins, ainsi que les libertés de ces libertés. Définir ainsi les ensembles complets de coups à essayer pour les jeux ip permet des gains de temps importants.

Une autre méthode proche pour la sélection des ensembles complets de coups possibles est l'utilisation de zones de relevance. Une zone de relevance à AtariGo est l'ensemble des intersections qui justifient une preuve. Lors d'une recherche, le programme peut mémoriser tous les tests effectués sur les intersections. L'ensemble ainsi constitué peut ensuite être traité afin d'obtenir avec certitude un ensemble complet d'intersections qui justifient la preuve. L'ensemble complet des coups forcés est alors l'ensemble des coups de Droite sur ces intersections et ce sont les seuls coups qui peuvent invalider la preuve. Cette méthode est similaire à la mémorisation d'une explication de la preuve et à son traitement pour trouver les ensembles complets de coups forcés [Cazenave 1996]. Elle est utilisée par la recherche lambda [Thomsen 2000] et par AGPS.

Un problème lié à l'utilisation de zones de relevance est qu'elles sélectionnent de nombreux coups inutiles, ce qui peut les rendre assez inefficaces. Par exemple, pour vérifier qu'une chaîne de pierres n'est pas capturée pendant une recherche, une liberté de cette chaîne doit être mémorisée dans la zone de relevance. Le coup correspondant sera testé dans les jeux d'ordre supérieur. Toutefois, si la chaîne a 6 libertés et que l'ordre du jeu est égal a 1 ou 2, il est clairement inutile de considérer la liberté comme coup possible. L'utilisation de connaissances abstraites sur le jeu permet alors d'éliminer de nombreux coups inutiles, et donc de réduire l'explosion combinatoire lors de la vérification des jeux combinatoires. Un exemple de connaissance abstraite très utile pour capturer les pierres au jeu de Go est qu'il n'est

pas possible de capturer une chaîne ennemie qui a n libertés en moins de n coups amis.
L'utilisation de connaissances abstraites sans zones de relevance est bien adaptée aux jeux combinatoires de la première section puisqu'on connaît à l'avance le nombre de coups maximum que Gauche peut jouer. Pour les jeux graduels, une combinaison de connaissances abstraites et de zones de relevance est appropriée.
Les connaissances abstraites sur les coups à envisager peuvent aussi être trouvées automatiquement en utilisant les règles du jeu exprimées en logique du premier ordre et un système de métaprogrammation comme Introspect [Cazenave, 1996,1998].

4 Algorithmes de recherche

Dans cette section, nous présentons les différents algorithmes de recherche liés à la recherche abstraite graduelle de preuve. Nous commençons avec le classique Alpha-Béta, puis nous décrivons la recherche abstraite de preuve, l'élargissement itératif, la recherche lambda et enfin la recherche abstraite graduelle de preuve.

4.1 L'Alpha-Béta

L'Alpha-Béta est l'algorithme de base pour résoudre les jeux à deux joueurs, à somme nulle et à information complète. De nombreuses améliorations ont été apportées à l'Alpha-Béta [Marsland & Björnsson 2000] parmi lesquelles l'approfondissement itératif, les tables de transpositions (une table de hachage des positions rencontrées dont le code de hachage est calculé très efficacement, et qui permet de ne pas recalculer plusieurs fois la même position a laquelle on peut arriver par des chemins différents) , la recherche de quiescence, la recherche avec fenêtre nulle et les heuristiques générales d'ordonnancement de coups comme l'heuristique de l'historique ou l'heuristique du coup qui tue.
Nous utilisons l'Alpha-Béta simple comme étalon pour mesurer les améliorations apportées par les autres algorithmes. La plupart des recherches sur l'Alpha-Béta ont été faites dans le cadre du jeu d'Echecs. Une particularité du jeu d'Echecs du point de vue des programmeurs est qu'il est plus rapide et moins risqué d'essayer tous les coups dans l'Alpha-Béta plutôt que d'essayer d'être sélectif, même si certains programmes utilisent l'heuristique du coup nul pour introduire parfois avec succès un peu de sélectivité. En dehors de l'heuristique de coup nul, peu de travaux ont été faits sur la sélectivité aux Echecs parce que le jeu ne s'y prête pas bien. En revanche, les jeux comme le jeu de Go, Hex ou le Phutball sont très sensibles aux algorithmes sélectifs du fait du très grand nombre de coups possibles dans ces jeux.

4.2 La Recherche Abstraite de Preuve

La recherche abstraite de preuve [Cazenave 2000] est basée sur les définitions des jeux combinatoires à valeurs inconnues données dans la section 2.1. Une recherche abstraite d'ordre n ne comprend pour ses nœuds ET que des coups associés à des jeux ip_k, avec $k \leq n$. Les jeux ip_k sont les jeux qui trouvent les coups forcés avec un arbre de profondeur 2k-1. Par exemple les jeux ip_3 sont trouvés en développant un arbre de profondeur 5, ils correspondent à la réunion des jeux graduels ip_{321} et ip_{311}.
A chaque nœud ET de la recherche abstraite de preuve, les différents jeux ip sont essayés en commençant par les jeux ip_1, puis ip_2 et ainsi de suite jusqu'au jeu ip_n, n étant l'ordre de la recherche. Si aucun jeu ip n'a été vérifié, le nœud ET est coupé et renvoie Perdu pour le joueur qui essaie d'atteindre le but (Gauche dans nos notations). Sinon, il essaie les coups associés au premier jeu $ip_{k,\ k \leq n}$ vérifié.
La recherche abstraite de preuve peut être vue comme un algorithme très sélectif de recherche, qui contrairement aux autres algorithmes sélectifs permet de rendre la recherche plus fiable, puisque les résultats que renvoie la recherche sont toujours justes. A chaque nœud ET de l'arbre de recherche, la vérification des jeux ip correspond à de petites recherches arborescentes avec approfondissement itératif.

4.3 L'Elargissement Itératif

L'élargissement itératif (IW) [Cazenave 2001] est une extension de la recherche abstraite de preuve. L'algorithme est basé sur la définition d'ensembles de coups abstraits qui peuvent être essayés aux nœuds ET de l'arbre de l'arbre à chaque seuil d'élargissement.

L'algorithme commence par une recherche abstraite de preuve avec des coups ip1, si cette recherche ne réussit pas, il essaie avec des coups ip1 et ip2, et ainsi de suite jusqu'à l'ordre maximum défini pour la recherche (en général 4).

Les résultats expérimentaux montrent que l'élargissement itératif permet de résoudre les problèmes approximativement deux fois plus vite qu'APS [Cazenave 2001].

4.4 La Recherche Lambda

La recherche Lambda a été conçue par T. Thomsen [Thomsen, 2000]. Elle peut être définie comme suit :

λ^n-arbre = Arbre contenant des λ^n-coups.

λ^n-coup = Si c'est le tour de Gauche, c'est un coup qui implique – si Droite passe – qu'il existe au moins un λ^i-arbre de valeur 1 (Gagné), $0 \leq i \leq n-1$. Si c'est le tour de Droite, c'est un coup qui implique qu'il n'existe pas de λ^i-arbre de valeur 1, $0 \leq i \leq n-1$.

La recherche lambda peut résoudre des problèmes très difficiles, mais peut aussi très rapidement dépasser les contraintes de temps, puisque chaque coup d'un λ^n-arbre doit être trouvé par un λ^{n-1}-arbre. Les zones de relevance

sont bien adaptées à la recherche lambda puisque les tailles des λ-arbres varient beaucoup, et ne sont pas connues à l'avance contrairement à la recherche abstraite de preuve.

4.5 AGPS

La recherche abstraite graduelle de preuve prend ses sources dans les trois algorithmes précédents, ainsi que dans la définition des jeux combinatoires graduels.

Elle consiste tout d'abord à ordonner les jeux combinatoires graduels par complexité croissante. Nous avons retenu l'ordre suivant qui suit approximativement un temps croissant de vérification des jeux : ip1, ip2, ip311, ip4111, ip51111, ip321, ip4121, ip4211, ip4221.

Pour un indice donné, on vérifiera dans l'arbre de recherche tous les jeux qui ont un indice inférieur à l'indice donné. Ainsi par exemple, une recherche d'indice 4 vérifiera pour tous les nœuds les jeux graduels ip1, ip2, ip311 et ip4111. Ces jeux sont testés aussi bien aux nœuds ET qu'aux nœuds OU de l'arborescence. Aux nœuds ET, si aucun jeu graduel n'est vérifié, le nœud est coupé et renvoie Perdu pour l'attaquant (l'attaquant est celui qui joue aux nœuds OU). Si un ou plusieurs jeux sont vérifiés, on essaie l'intersection des ensembles de coups associés aux jeux vérifiés. Aux nœuds OU, les jeux graduels sont testés pour l'attaquant (on détecte alors les coups forcés de l'attaquant pour ne pas perdre), si aucun des jeux n'est vérifié, contrairement aux nœuds ET, on essaie alors tous les coups possibles.

Pour un indice donné, l'algorithme continue sa recherche aussi profondément que la vérification des jeux le permet. Lorsque la recherche renvoie Perdu, on passe à l'indice suivant et on reconstruit un nouvel arbre de recherche.

On peut voir sur la figure 7 l'intérêt des jeux graduels. Le coup numéro 2 de la séquence est trouvé très rapidement par un jeu graduel ip_{4111}, qui est plus simple à vérifier qu'un jeu ip_{321} mais d'un ordre plus grand. La recherche abstraite de preuve aurait besoin d'une recherche d'ordre 4 avant de trouver ce premier coup et chercherait à vérifier les jeux ip_4 à tous les nœuds ET, ce qui est très coûteux. De même, AGPS permet une distinction plus fine entre les jeux que la recherche Lambda qui doit développer un λ^3-arbre avant de trouver le coup 6 de la figure 7, alors qu'il est trouvé par un jeu ip_{4221} par AGPS, et donc ne trouve la solution d'AtariGo qu'avec un λ^4-arbre, et en testant beaucoup de coups inutiles dans ses zones de relevance, alors que les zones de relevance abstraites de AGPS sont plus sélectives.

5 Résultats Expérimentaux

Nous présentons ici l'application d'AGPS à AtariGo, et nous comparons son efficacité à l'Alpha-Béta et à APS dans divers problèmes d'AtariGo que nous avons créés pour l'occasion. La conséquence de la résolution d'AtariGo par AGPS est que nous disposons désormais d'un joueur parfait d'AtariGo qui joue tous ses coups quasi-instantanément. Ceci est possible en mémorisant la table de transposition après avoir trouvé la solution.

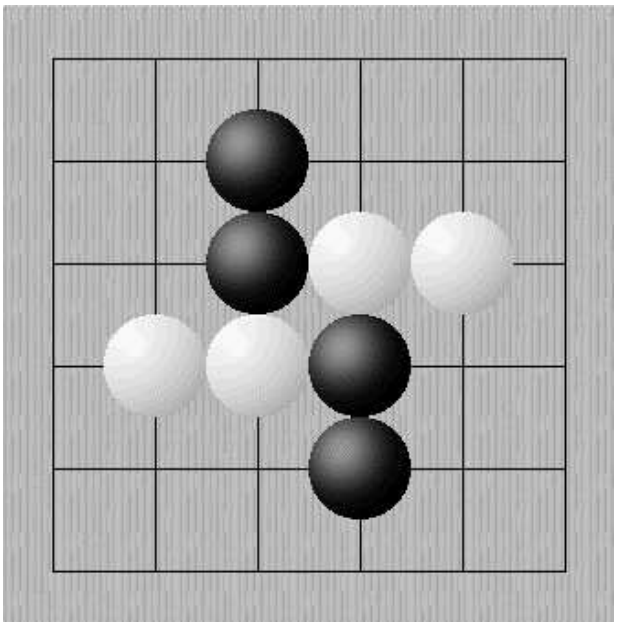

Figure 4. Problème 1 d'AtariGo 6x6 : ordre 4 et profondeur 13

P	Alpha-Béta		IW_4		$AGPS_{4221}$	
	T (s)	Nds	T (s)	Nds	T (s)	Nds
2	0.00	59	13.4	118	0.78	256
3	0.01	787	14.1	274	0.79	360
4	0.01	1805	96.7	354	3.71	384
5	0.09	18893	133	2346	4.83	579
6	0.49	86495	569	3043	**4.69**	**362**
7	2.77	566421	**284**	**6216**	**5.81**	**368**
8	29.2	5233926	**961**	**7825**	**8.36**	**411**
9	142	28239207	**1517**	**55929**	**8.55**	**477**
10	2035	383263757			**9.48**	**471**
11	9038	1701395805				

Tableau 1. Comparaison des algorithmes sur le problème 1

Les données en gras dans le tableau correspondent aux temps et aux profondeurs pour lesquelles le problème a été résolu. L'Alpha-Béta doit effectuer une recherche à profondeur 13 pour résoudre le problème, aucune des profondeurs testées ne correspond donc à une solution. Si on extrapole les données du tableau 1, on peut estimer le temps nécessaire à la résolution du problème de la figure 4 par Alpha-Béta à plusieurs jours. Les algorithmes APS et AGPS trouvent la solution à une profondeur inférieure à 13 parce qu'ils se rendent compte à cette profondeur inférieure qu'une position vérifie un jeu ip qui a un ensemble de coups forcés associé vide, ce qui rend la position gagnée pour l'adversaire (on peut noter qu'un jeu d'ordre 4 correspond à une recherche à profondeur 8, il

est donc normal que la solution soit trouvée à une profondeur inférieure de 7 à la profondeur de la solution la plus courte).
On peut noter que les temps mis par AGPS comprennent les parcours d'arbre avec tous les jeux graduels d'ordre inférieurs à l'ordre du jeu solution. Ainsi, la moitié du temps de AGPS est consommé par des parcours qui n'amènent pas de solutions avec des jeux d'indices graduels inférieurs à l'indice graduel de la solution. Le parcours d'arbre des jeux ip321 notamment consomme beaucoup de temps sans pouvoir trouver la solution de ce problème.

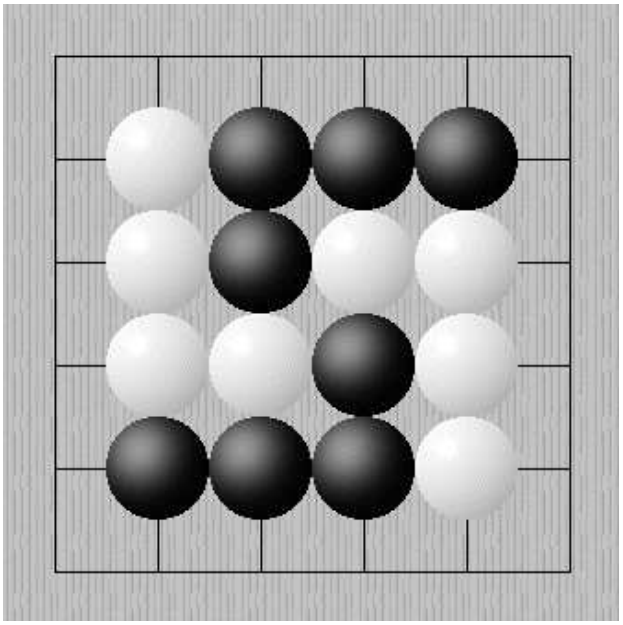

Figure 5. Problème 2 : ordre 3 et profondeur 15

P	Alpha-Béta		IW_3		$AGPS_{4221}$	
	T (s)	Nds	T (s)	Nds	T (s)	Nds
2	0.00	43	0.07	64	0.75	200
3	0.00	403	0.08	136	0.74	346
4	0.01	791	0.27	168	2.04	458
5	0.03	5783	0.31	531	2.59	881
6	0.19	28333	0.66	601	3.96	988
7	0.74	133197	0.78	1579	4.42	1628
8	10.00	1608818	1.42	1779	**4.60**	**1481**
9	33.94	5457041	1.18	3286	**4.91**	**2363**
10	200.6	30054517	**1.74**	**3629**	**1.19**	**686**
11	456.3	77884056				

Tableau 2. Comparaison des algorithmes sur le probleme 2

Dans les tests des tableaux 1, 2 et 3 nous avons artificiellement limité la profondeur de recherche d'AGPS, et nous l'avons aussi limité aux jeux ip_{4221}. La limitation artificielle de profondeur lui fait perdre beaucoup de temps par rapport à l'algorithme normal qui approfondit une recherche avec un jeu donné jusqu'à ce qu'il trouve la solution ou épuise les possibilités. On voit bien que pour la profondeur solution du tableau 2 (qui est de 8 pour AGPS), le temps de trouver la solution est bien plus court que celui de l'Alpha-Béta (qui devrait aller jusqu'à la profondeur 15 pour la trouver), mais aussi que le temps de résolution à profondeur 10 est plus court que celui d'APS à même profondeur sur un problème qui est d'ordre assez faible pour être résolu efficacement par APS (APS le résout en 1.74s à profondeur 10). La solution trouvée par AGPS à profondeur 8 est plus coûteuse en temps mais trouve une solution plus courte au problème. On voit ici le handicap que constitue la fixation d'une profondeur maximum pour AGPS. Il est clair que si l'on utilise l'algorithme AGPS normal qui consiste à approfondir tant qu'on peut pour un jeu donné, il trouvera la solution à profondeur 10 et s'arrêtera aux jeux ip321. Il la trouvera en 1.19s ce qui est légèrement plus rapide qu'APS parce qu'il utilise les coups forcés aux niveaux OU aussi, ce qui est une amélioration par rapport à APS.

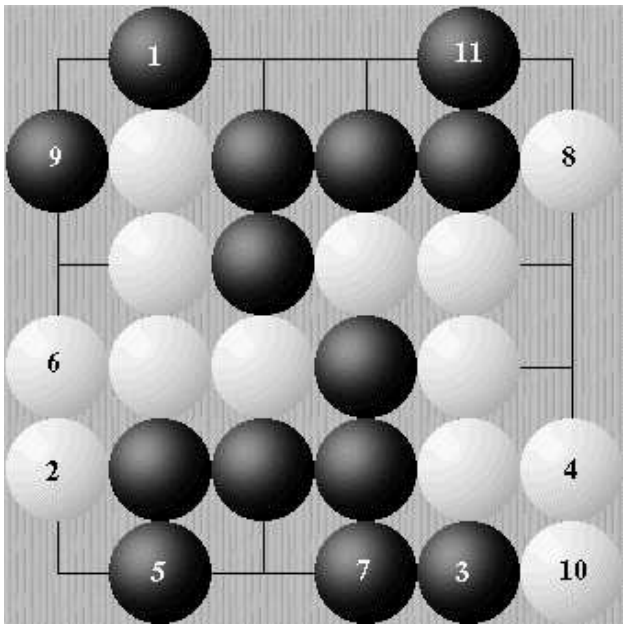

Figure 6. Solution trouvée par AGPS au problème 2

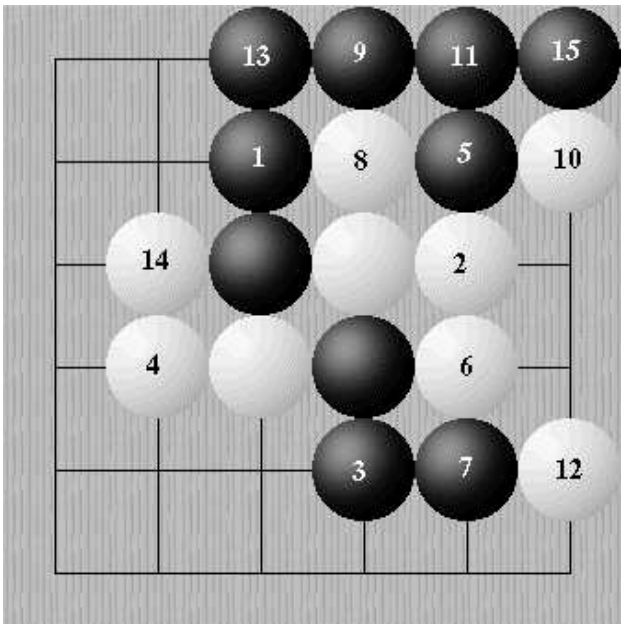

Figure 7. La séquence de gain la plus courte trouvée par AGPS pour AtariGo 6x6

Profondeur	$AGPS_{4221}$	
	Temps	Nœuds
2	9.80s	384
3	16.46s	1484
4	29.65s	1870
5	45.64s	5359
6	78.76s	5831
7	95.86s	9352
8	247.66s	11346
9	304.74s	19079
10	**332.94s**	**15130**

Tableau 3. AGPS pour la résolution d'AtariGo 6x6

Le tableau 3 donne le nombre de nœuds et le temps mis pour parcourir l'arbre de jeu en utilisant AGPS avec différentes limites de profondeur. Les tables de transpositions n'ont pas été utilisées dans cette expérience : pour chaque profondeur le programme part de zéro, sans aucune information venant des profondeurs inférieures. Ceci dans le but de donner une idée de l'évolution de la complexité d'AGPS avec la profondeur d'un problème à résoudre. On voit ainsi que le nombre de nœuds et le temps de parcours augmentent très lentement avec la profondeur du problème, contrairement à l'Alpha-Béta qui comme on l'a vu précédemment et comme le montre son analyse théorique augmente exponentiellement avec un facteur de l'ordre de la racine du nombre de coups possibles dans le meilleur cas (nos résultats expérimentaux tendent à montrer qu'un niveau de profondeur de plus revient à multiplier par à peu près 6 le temps et le nombre de nœuds du parcours d'arbre par Alpha-Béta dans le cas d'AtariGo 6x6).
Si on extrapole les données du tableau 1 pour l'Alpha-Béta, étant donné que la profondeur de résolution d'AtariGo 6x6 est de 17, on arrive à une durée de plusieurs années pour la résolution d'AtariGo 6x6 par Alpha-Béta.

Jeu	Appels	Temps	Temps/Appel
ip1	3371005	47820000	14
ip2	282408	117270000	415
ip311	4902	150000	30
ip4111	4510	240000	53
ip51111	4118	180000	43
ip321	3726	15890000	4264
ip4121	3334	880000	263
ip4211	2942	14860000	5050
ip4221	3403	92880000	27293

Tableau 4. Temps par jeu pour la résolution d'AtariGo 6x6

Le tableau 4 donne les temps passés dans la vérification de chaque jeu ip pour la résolution d'AtariGo 6x6. Il est intéressant de noter, que le jeu d'AtariGo rend impossible par définition la vérification des jeux graduels ip4121 et ip4211, toutefois, ces jeux ont tout de même été inclus dans AGPS pour AtariGo, de façon à garder toute sa généralité à l'algorithme. On peut voir que ip4211 est assez consommateur de ressources inutiles. On pourrait encore gagner du temps sur la résolution en l'éliminant des jeux à vérifier. Il est aussi intéressant d'étudier pour AtariGo et pour d'autres jeux l'ensemble des jeux graduels qui ne peuvent jamais être vérifiés, et de le trouver si possible de manière automatique.

6 Conclusion et Extensions Possibles

Nous avons présenté les jeux combinatoires graduels à valeurs inconnues. Nous avons aussi présenté AGPS, un nouvel algorithme de recherche d'arbres ET/OU basé sur les jeux combinatoires graduels. AGPS a une efficacité accrue de plusieurs ordres de grandeurs sur Alpha-Béta, il est aussi beaucoup plus efficace qu'APS et la recherche Lambda qui sont des algorithmes récents spécialisés dans ce type de problèmes. L'illustration de cette efficacité a été faite avec AtariGo, un jeu dérivé du jeu de Go qui est utilisé pour apprendre le Go aux joueurs débutants.
AGPS peut bien entendu être utilisée pour améliorer la rapidité de résolution de jeux déjà résolus comme le Gomoku, le Renju, puissance 4, Hex pour de petits damiers (7x7 par exemple qui est la limite actuelle de résolution) ou le Football des Philosophes qui est pour

l'instant résolu jusqu'à la taille 11x11 par APS, ainsi que pour les résoudre pour de plus grandes tailles.
Elle peut aussi permettre de résoudre de nouveaux jeux comme AtariGo que nous avons présenté, mais aussi d'améliorer les programmes de jeux existants en leur permettant de résoudre des problèmes qui étaient jusqu'à présent hors de leur portée. On peut ainsi améliorer les calculs de prise, de connexion, de vie et de mort au jeu de Go, les calculs de connexions à Hex, les éjections à Abalone, les mats aux Echecs, et ainsi de suite…
Une application directe de la résolution d'AtariGo est la mise à disposition d'un bon joueur informatique qui peut aider les grands débutants à maîtriser les bases du jeu de Go.
Une extension possible de cette recherche serait de réutiliser les idées que nous avons utilisées dans les jeux pour améliorer des programmes dans des domaines connexes qui utilisent aussi des arbres ET/OU comme par exemple la démonstration automatique de théorèmes mathématiques. Des parallèles similaires ont déjà été effectués avec sucés comme par exemple l'algorithme d'élargissement itératif qui est analogue à des algorithmes qui permettent d'améliorer la résolution de contraintes [Cazenave 2001].
La théorie de la complexité des jeux qui sous-tend AGPS mérite d'être testée du point de vue des sciences cognitives. Le jeu graduel maximum pour résoudre un problème ainsi que la distribution des jeux graduels utilisés dans la séquence principale semblent être de bons indicateurs de la difficulté intrinsèque d'un problème. Il est fort probable que cette difficulté théorique soit fortement corrélée à la difficulté réelle du problème pour un joueur. Une extension prévue de notre travail est l'évaluation expérimentale de cette mesure en vérifiant qu'elle rend bien compte de la difficulté réelle du problème. Une théorie valide de la difficulté des problèmes permettrait entre autre d'engendrer automatiquement des problèmes intéressants pour un joueur d'un niveau donné, et ainsi de construire un programme qui adapterait automatiquement ses leçons au niveau de son élève.

Bibliographie

[1] Cazenave T.: *Système d'Apprentissage par Auto-Observation. Application au Jeu de Go*. Thèse de Doctorat, Université Paris 6. 1996.
[2] Cazenave T.: *Metaprogramming Forced Moves*. Proceedings ECAI98 pp. 645-649, Brigthon, 1998.
[3] Cazenave T.: *Abstract Proof Search*. Computers and Games 2000. To appear in LNCS 2001.
[4] Cazenave T.: *Iterative Widening*. Proceedings of IJCAI-01, Vol. 1, pp. 523-528, Seattle, 2001.
[5] Conway J., Berlekamp E., Guy R.: *Winning ways*, Tome 1 and 2, Academic Press, 1982.
[6] Marsland T. A., Björnsson Y.: *From Minimax to Manhattan.* Games in AI Research, pp. 5-17. Edited by H.J. van den Herik and H. Iida, Universiteit Maastricht. ISBN 90-621-6416-1. 2000.
[7] Pitrat J.: Realization of a general game-playing program. Information Processing 68. North Holland Publishing Company, Amsterdam, pp. 1570-1574, 1969.
[8] Pitrat J.: Games: The Next Challenge. in: ICGA Journal, Vol. 21, N° 3, September 1998, pp. 157-156. 1998.
[9] Thomsen T.: *Lambda-search in game trees – with application to go*. Computers and Games 2000. To appear in LNCS 2001.