HAL open science

Du jeu de Go au Havannah : variantes d'UCT et coups décisifs

Fabien Teytaud, Olivier Teytaud

▶ **To cite this version:**

Fabien Teytaud, Olivier Teytaud. Du jeu de Go au Havannah : variantes d'UCT et coups décisifs. RFIA, Jan 2010, Caen, France. ⟨inria-00454541⟩

HAL Id: inria-00454541

https://inria.hal.science/inria-00454541v1

Submitted on 8 Feb 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Autorisation HAL

HAL Authorization

Du jeu de Go au Havannah : variantes d'UCT et coups décisifs

F. Teytaud and O. Teytaud
TAO (Inria), LRI, UMR 8623(CNRS - Univ. Paris-Sud),
bat 490 Univ. Paris-Sud 91405 Orsay, France, fteytaud@lri.fr

30 novembre 2009

Résumé

Les algorithmes de type fouille d'arbre Monte-Carlo et UCT (upper confidence tree) ont révolutionné le jeu de Go par ordinateur depuis 2006/2007. Quelques applications, encore rares, ont montré la généralité de ces approches, en particulier quand l'espace d'actions est trop grand pour les autres techniques, et quand l'état est complètement observable. Dans ce papier, nous testons cette généralité, en expérimentant UCT dans un autre jeu, le Havannah. Ce jeu est connu spécialement difficile pour les ordinateurs. Nous montrons que cette approche donne de bons résultats tout comme pour le jeu de Go, même si on peut noter quelques différences et en particulier la notion de coup décisif, inexistante en Go.

Remarque : Une version préliminaire de ce travail a été publiée en anglais dans la conférence ACG12.

Mots clef
UCT, Fouille d'Arbre Monte-Carlo, Havannah, coups décisifs.

Abstract
Monte-Carlo Tree Search and Upper Confidence Bounds provided huge improvements in computer-Go. In this paper, we test the generality of the approach by experimenting on another game, Havannah, which is known for being especially difficult for computers. We show that the same results hold, with slight differences related to the absence of clearly known patterns for the game of Havannah, in spite of the fact that Havannah is more related to connection games like Hex than to territory games like Go. However, for optimal performance in Havannah we must take care of decisive moves, a notion which does not make sense in the case of Go.

Keywords
UCT, Monte-Carlo Tree Search, Havannah, decisive moves.

1 Introduction

Le jeu de Havannah, est un jeu de plateau à deux joueurs inventé par Christian Freeling [20]. Ce jeu est joué sur un plateau hexagonal, avec des cases hexagonales. La taille d'un plateau peut varier, mais elle est en générale pour deux bons joueurs de 8 ou 10. Chaque joueur joue à tour de rôle, le joueur blanc commence (le second joueur possède donc des pierres noires). Les règles sont simples :

- Chaque joueur joue en posant une pierre dans un emplacement libre. S'il n'y a plus de case libre et qu'aucun joueur n'a gagné alors il y a égalité. Ces cas sont très rares.
- Pour gagner, un joueur doit réaliser l'une de ces trois formes :
  - Un anneau, c'est à dire une boucle autour d'une où de plusieurs cases (les cases peuvent être occupées par des pierres blanche, noires, ou des emplacements vides).
  - Un pont, c'est à dire, une chaîne continue de pierres reliant un des six coins à un autre.
  - Une fourche, c'est à dire une chaîne continue, connectée à trois côtés. Les coins sont considérés comme n'appartenant à aucun côté.

Ces trois formes de victoire sont présentées en Figure 1.

Sur une grande taille de plateau, les meilleurs programmes de Havannah ont un niveau très faible par rapport aux joueurs humains. En 2002, Christian Freeling a offert un prix de 1000 euros, valable jusqu'en 2012, pour le premier programme qui serait capable de le battre au moins une partie sur une série de dix matchs.

Le jeu de Havannah peut être vu comme étant à la frontière entre le Go et le Hex. C'est un jeu entièrement observable, impliquant des connexions ; le concept de boucle se rapproche plus du jeu de Go, cela pouvant être vu comme le concept des yeux ou de capture dans ce jeu. Le jeu de Go a longtemps été le principal défi pour les intelligences artificielles dans les jeux, car c'est un jeu très populaire, notamment dans l'ensemble des pays asiatiques. Toutefois, on ne peut plus dire que les programmes aient un niveau débutant. Depuis les premiers programmes implémentant les approches MCTS/UCT [6, 9, 14], de nombreuses améliorations sont apparues, comme "First-Play Urgency" [19], "RAVE" [4, 12], motifs et "progressive widening" [10, 7], qui sont meilleurs que le terme d'exploration de la formule UCB (Upper confidence Bounds),

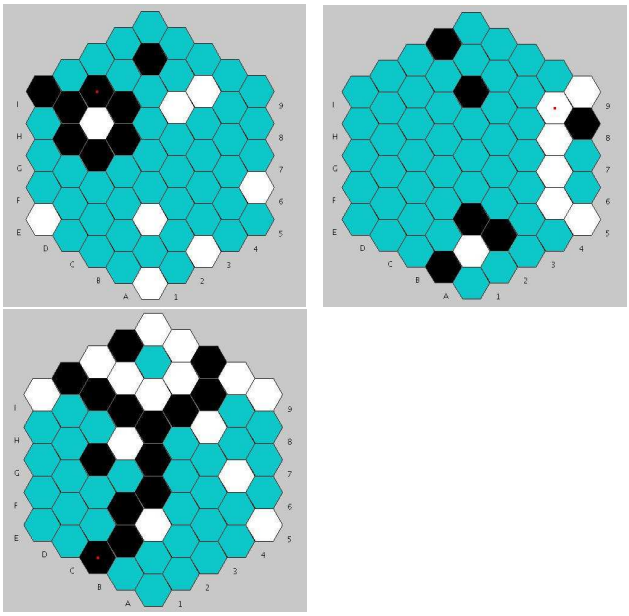


FIG. 1 – Trois parties finies : en haut à gauche, un anneau (une boucle faite par noir), en haut à droite, un pont, qui relie deux côtés, réalisé par blanc, et enfin, en bas, une fourche, qui relie trois côtés pour le joueur noir.

parallélisation à grande échelle [11, 8, 5, 13], construction automatique de bibliothèques d'ouvertures [2]. Grâce à l'ensemble de ces améliorations, le programme MoGo a déjà gagné des parties de Go en taille 9x9 contre un joueur professionnel (Amsterdam, 2007 ; Paris, 2008 ; Taiwan 2009), mais a également récemment gagné contre un joueur professionnel, Li-Chen Chien (Tainan, 2009), en taille 19x19, avec un handicap de 6 pierres, ainsi que contre Zhou Junxun, un joueur 9P "top pro" (ayant gagné un des tournois majeurs de Go), gagnant de la coupe LG 2007 (Tainan, 2009), avec un handicap de 7 pierres.

Les caractéristiques suivantes font que le jeu de Havannah est un jeu très difficile pour les ordinateurs, au même titre que le jeu de Go :

- peu de patterns sont connus pour le jeu de Havannah ;
- aucune fonction d'évaluation naturelle ;
- aucune règle permettant un élagage de l'ensemble des coups possibles ;
- un grand espace d'actions (271 pour le premier coups d'un plateau de taille 10).

L'avantage du jeu de Havannah, tout comme le jeu de Go (à l'exception de quelques cas pathologiques) est que les simulations ont une taille limitée par la taille du plateau.

Le but de ce papier est de s'intéresser à la généralité de cette approche en la testant pour le jeu de Havannah.

Pour l'ensemble du papier, nous définisssons la notation $x \pm y$ comme étant un résultat de moyenne x, et un intervalle de confiance de 95% $[x - y, x + y]$ (*i.e.* y est deux écarts types).

2 UCT

"Upper confidence Trees" est un cas particulier d'algorithme MCTS, et est le choix le plus naturel avant optimisations spécifiques du problème ; c'est donc UCT que l'on présentera ici comme version de base. Le principe de base est comme suit. Tant qu'il reste du temps pour jouer, l'algorithme effectue une simulation dans l'arbre UCT, jusqu'à une feuille de cet arbre ; puis poursuit la simulation depuis une feuille de l'arbre UCT jusqu'à une fin de partie. Ces simulations sont des parties complètes, de la racine de l'arbre jusqu'à ce que la partie soit terminée, et il convient de spécifier quatre choses :

- *Quel coup sera finalement joué, quand le temps disponible est écoulé (il ne s'agit plus de simuler des coups, mais de choisir le coup réellement joué).* Une heuristique fiable consiste à jouer le coup ayant conduit au plus grand nombre (et non au plus grand taux) de victoires ; c'est la solution ici retenue.
- *Comment sont jouées les simulations dans l'arbre.* Lorsque l'on est dans une situation s qui est déjà dans l'arbre UCT, alors les différents choix dépendent des statistiques suivantes : nombre de victoires et nombre de défaites pour les parties précédentes, pour chaque coups légaux de la situation s. Une formule choisissant un coup dépendant des statistiques est appelée "une formule de bandit" ; la formule de bandit classique UCT sera détaillée dans la section suivante.
- *Comment sont jouées les simulations en dehors de l'arbre.* Dans la version la plus basique, pour ce qui est des simulations en dehors de l'arbre, un joueur aléatoire, lorsqu'une situation s qui n'est pas dans l'arbre apparait, joue simplement des coups au hasard et uniformément parmi les coups légaux de la situation s ; nous n'avons pas implémenté autre chose de plus sophistiqué (à part les coups décisifs évoqués plus bas).
- *Enfin, comment l'arbre UCT grandit.* Après chaque simulation, la première situation d'une partie simulée qui n'était pas encore en mémoire est archivée, et toutes les statistiques concernant le nombre de victoires et de défaites sont mis à jour dans chaque situation en mémoire qui a été traversée par la simulation.

Formule de bandit

La formule de bandit la plus classique est UCB (Upper Confidence Bounds [15, 3]). Les méthodes de type fouille d'arbre Monte-Carlo basées sur UCB sont appelées UCT. L'idée est de calculer un score d'exploration/exploitation pour chaque coup légal dans une situation s pour choisir quel coup doit être simulé ; le

coup avec le score le plus grand sera choisi.

Le score d'exploration/exploitation d'un coup d est typiquement la somme du taux de succès empirique $score(d)$ du coup d, et d'un terme d'exploration qui est plus élevé pour les coups les moins explorés. UCB utilise la borne de Hoeffding pour définir le terme d'exploration :

$$exploration_{Hoeffding} = \sqrt{\log(2/\delta)/n}$$

avec n le nombre de simulations pour le coup d, δ étant en général choisi linéairement en fonction du nombre m de simulations pour la situation s. Pour notre implémentation, avec un joueur aléatoire uniforme, la formule a empiriquement été fixée à :

$$exploration_{Hoeffding} = \sqrt{0.25 \log(2+m)/n}. \quad (1)$$

[1, 17] a montré que l'utilisation de la borne de Bernstein à la place de la borne de Hoeffding pouvait être plus efficace. Dans ce cas, le terme d'exploration devient :

$$exploration_{Bernstein} = \sqrt{score(d)(1-score(d))2\log(3/\delta)/n} + 3\log(3/\delta)/n \quad (2)$$

Ce terme est plus petit pour les coups avec une variance faible, et ceci quel que soit $score(d)$.

Après le paramètrage empirique de l'équation de Hoeffding (Eq. 1), nous avons testé la formule de Bernstein suivante (avec $p = score(d)$) :

$$exploration_{Bernstein} = \sqrt{4Kp(1-p)\log(2+m)/n} + 3\sqrt{2}K\log(2+m)/n. \quad (3)$$

A l'exception du "2+", qui est ajouté pour éviter les cas spéciaux pour 0, il s'agit de la formule de Bernstein pour δ linéaire en fonction de m.

Nous avons testé différentes valeurs de K. La première valeur 0.25 correspond à la borne de Hoeffding excepté que le second terme est ajouté.

K	Score contre la formule de Hoeffding Eq. 1
0.250	0.503 ± 0.011
0.100	0.578 ± 0.010
0.030	0.646 ± 0.005
0.010	0.652 ± 0.006
0.001	0.582 ± 0.012
0.000	0.439 ± 0.015

Ces expériences ont été faites avec 1000 simulations par coup, sur un plateau de taille 5. Nous avons essayé des valeurs pour la constante K inférieures à 0.01 mais nous n'avons obtenu aucun bon résultat.

Passage à l'échelle d'UCT

En général pour l'algorithme UCT, plus le nombre de simulations est grand, meilleurs sont les résultats. Typiquement, le taux de victoires avec $2z$ simulations contre UCT avec z simulations est de 63% dans le cas du jeu de Go [11].

Dans le cas du Havannah, avec un joueur aléatoire uniforme (c'est à dire que le joueur joue uniformément parmi les coups légaux) et la formule de bandit définie à l'équation 1, nous obtenons les résultats suivant pour UCT paramétré comme précédemment ($K = 0.25$) :

Nombre de simulations des deux joueurs	Taux de succès
250 vs 125	0.75 ± 0.02
500 vs 250	0.84 ± 0.02
1000 vs 500	0.72 ± 0.03
2000 vs 1000	0.75 ± 0.02
4000 vs 2000	0.74 ± 0.05

Ces expériences ont été faites sur un plateau de taille 8.

3 Guider l'exploration

Les algorithmes de type UCT sont bons dans le compromis entre exploration et exploitation. Toutefois, ils ne fournissent aucune information sur les coups non explorés, et sur comment choisir parmi ces coups (le terme d'exploration est le même pour tous les coups non explorés, et le terme d'exploitation n'est pas défini) ; et que peu d'information pour les coups faiblement explorés. Différentes astuces ont été proposées autour de cette idée : First Play Urgency, Progressive widening, et Rapid Action Value Estimates (RAVE, estimateur rapide de valeurs d'actions).

3.1 Progressive widening/unpruning

Avec le progressive widening [10, 7, 18], nous débutons par classer les coups légaux d'une situation s selon une heuristique : les coups sont renommés 1, 2, ..., n, avec $i < j$ si le coup i est préféré au coup j pour cette heuristique. Donc, à la m-ième simulation d'un noeud, tous les coups avec un index plus grand que $f(m)$ ont un score $-\infty$ (c'est à dire qu'ils sont écartés), avec $f(m)$ une application croissante de $\mathbb{N}$ dans $\mathbb{N}$.

Il a été montré dans [18] que cette amélioration fonctionne également avec un classement aléatoire, avec *e.g.* $f(m) = \lfloor Km^{1/4} \rfloor$ pour une certaine constante K. Dans [10] il a été montré que $f(m) = \lfloor Km^{1/3.4} \rfloor$ pour un certain K fonctionne bien dans le cadre du Go, avec une heuristique basée sur les patterns. L'algorithme proposé dans [7] et qui est maintenant utilisé dans MoGo, est légèrement différent : un terme d'exploration dépendant d'une heuristique basée sur les patterns et qui décroit logarithmiquement avec le nombre de simulations de ce coup est ajouté au score ;

pour le coup d dans la situation s,

$$nouveauScore(d,s) = score(d,s) + H(d,s)/\log(2+m),$$

avec comme précédemment n le nombre de simulations incluant le coup d à la situation s. Dans le cas du Havannah, nous n'avons aucune heuristique de cette sorte. Nous avons décidé d'utiliser le progessive widening sans heuristique (avec un classement aléatoire), comme il a été montré dans [18] qu'une amélioration est possible même sans heuristique. L'idée d'utiliser le progressive widening sans heuristique est un peu contre intuitive. Toutefois considérons le cas suivant. Considérons un noeud d'un arbre qui est exploré 50 fois seulement (ceci arrive certainement pour certains noeuds de l'arbre). S'il y a 50 coups légaux à partir de ce noeud, alors les 50 simulations vont être réparties pour l'ensemble des 50 coups légaux (une simulation par coup légal) dans le cas où le progressive widening n'est pas utilisé. Dans le cas contraire, le progressive widening va essayer quelques coups, par exemple 4 coups, et beaucoup plus essayer le meilleur de ces 4 coups - ceci est certainement meilleur que de prendre la moyenne de tous les coups comme fonction d'évaluation. Nous avons expérimenté ceci avec 500 simulations par coup pour un plateau de taille 5, et différentes valeurs pour les constantes P et Q pour le progressive widening $f(m) = Q\lfloor m^P \rfloor$:

Q, P	Taux de succès contre UCT sans progressive widening
1, 0.7	0,496986 ± 0,014942
1, 0.8	0.51 ± 0,0235833
1, 0.9	0.50 ± 0,0145172
4, 0.4	0,500454 ± 0,0134803
4, 0.7	0.49 ± 0,0181818
4, 0.9	0,485101 ± 0,0172619

Ces expériences ont été faites avec le terme d'exploration donné par l'équation 1. Nous avons testé d'autres paramètres pour P et Q sans obtenir d'amélioration significative. Peut-être que des améliorations sont possibles en paramétrant la formule de Hoeffding et la formule de progressive widening.

3.2 Estimateur rapide de valeur d'action

Pour le cas du Go, [4, 12] proposent de moyenner le score avec un estimateur à base de permutation. Cet estimateur à base de permutation est appelé "RAVE" pour "rapid action-value estimate". Précisément, le score pour un coup d dans une situation s simulée n fois devient :

$$nouveauScore(d,s) =$$

$$(1-\alpha(n))score(d,s) + \alpha(n)rave(d,s) + exploration$$

avec

- $score(d,s)$ le ratio des simulations gagnées avec la situation s et le coup d dans s, $rave(d,s)$ est la proportion de parties gagnées parmi les parties *contenant* le coup d après la situation s.
- $\alpha(n) \to 0$ quand $n \to \infty$, alors que $\alpha(n)$ est proche 1 pour n petit, donc les valeurs de l'heuristique *rave* sont utilisés initialement, s sont eventuellement remplacées par les valeurs "réelles".

La différence entre $score(d,s)$ et $rave(d,s)$ est que la proportion de parties gagnées dans $rave()$ est calculée parmi toutes les situations *contenant d comme coup après la situation s* et pas seulement dans toutes les simulations *avec d comme coup à la situation s*. Dans le jeu de Go, les valeurs RAVE apportent une grande amélioration. Toutefois, cela implique une implémentation compliquée à cause du phénomène des captures et re-captures. Dans le cas du jeu de Havannah, ce problème n'existe pas, et nous allons voir que les résultats sont très bons. Nous avons utilisé cette formule :

$$\alpha(n) = R/(R+n),$$

$$exploration =$$

$$exploration_{Hoeffding} = \sqrt{K\log(2+m)/n}.$$

R a été empiriquement fixé à 50 (sur les parties avec une taille de plateau de 5 avec 1000 simulations par coup) ; intuitivement R est le nombre de simulations à effectuer avant que le poids des valeurs "RAVE" ($rave(d,s)$) soit égal au poids des valeurs "réelles" ($score(d,s)$). Tous les résultats suivants sont effectués avec le terme d'exploration donné par l'Eq. 1.

Taille 5	R=50, K=0.25, taille 5, 1000 simulations/coup	47.26% ± 4.0 %
	R=50, K=0.05, taille 5, 1000 simulations/coup	60.46% % ± 2.9 %
	R=50, K=0, taille 5, 1000 simulations/coup	95.33% ± 0.01 %
Taille 8	R=50, K=0, taille 8, 1000 simulations/coup	100% on 1347 runs

$K = 0$ est la meilleure constante. Ceci a également été constaté dans [16] pour le jeu de Go. Nous avons ensuite testé avec un nombre de simulations plus important, 30 000 simulations par coup. Malheureusement mais en accord avec [16], nous devons modifier les coefficients pour avoir des résultats positifs, alors que le paramètrage d'UCT semble être plus indépendant du nombre de simulations par coup.

R=50, K=0	0.53 ± 0.02
R=50, K=0.25	0.47 ± 0.04
R=50, K=0.05	0.60 ± 0.02
R=50, K=0.02	0.60 ± 0.03
R=5, K=0.02	0.61 ± 0.06
R=20, K=0.02	0.66 ± 0.03

Tous ces résultats correspondent à une taille de plateau de 5, et 30 000 simulations par coup. La première ligne correspond à la configuration empirique choisie pour 1000 simulations par coup ; ces résultats sont décevants, nous ne sommes qu'à peine meilleur qu'UCT avec 30 000 simulations par coup. La seconde ligne utilise la même constante d'exploration que UCT, mais il semble que cela soit trop. Puis, pour les lignes suivantes, en utilisant un terme d'exploration plus faible ainsi que des valeurs de R également plus faibles, nous avons de bons résultats. [16] remarque que, avec un grand nombre de simulations, $K = 0$ est le meilleur choix, mais un bonus d'exploration dépendant de motifs locaux était utilisé à la place.

Essayons de synthétiser nos résultats pour de grands nombres de simulations. Dans ce contexte, RAVE n'est pas aussi efficace que pour de petits nombres de simulations (en comparaison avec UCT - il reste toutefois significativement meilleur, mais de peu). Peut-être qu'un meilleur paramétrage pourrait donner de meilleurs résultats. Les principales faiblesses de RAVE sont ainsi selon nous :

- un paramètrage spécifique est nécessaire pour différents nombres de simulations
- les résultats pour de grands nombres de simulations sont moins impressionnants (mais significatifs).

D'un autre côté, l'efficacité de RAVE augmente avec la taille de l'espace d'actions - ceci est prometteur pour l'application de RAVE pour les grands espaces d'actions (par exemple, une taille de plateau de 10).

4 Parties contre Havannah-Applet

Nous avons testé notre programme contre "Havannah-Applet" `http://dfa.imn.htwk-leipzig.de/havannah/`, recommandé par l'association Mind-Sports. L'applet joue en 30 secondes pour chacun de ses coups, mais nous avons choisi de restreindre notre programme à 8 secondes par coup pour la première partie et 2.5 secondes par coup pour la seconde. Dans les deux cas, notre programme, basé sur RAVE, sans terme d'exploration, pas de progressive widening, était noir (les blancs commencent et ont par conséquent un avantage). La première partie (jouée avec 8 secondes par coup pour notre programme, contre 30 secondes pour l'adversaire) est présentée en Fig. 2. Le taux de succès est en dessous de 50% au début du jeu (car l'adversaire jouant en premier a un avantage initial). Il augmente ensuite régulièrement jusqu'à la fin de partie. La seconde partie est présentée en Fig. 3, avec seulement 7000 simulations par coup. Encore une fois, et pour la même raison, le taux de succès est inférieur de 50% au début du jeu. Là encore, le taux de succès augmente régulièrement jusqu'à la fin de partie.

5 Coups décisifs : différences avec le jeu de Go

Au jeu de Go, il n'existe pas de coup décisif terminant la partie, puisque la victoire est évaluée sur une question de territoire. Au Havannah par contre, la partie s'arrête lorsque l'on pose la pierre qui termine un anneau, un pont ou une fourche. On peut donc très facilement modifier UCT pour que :

- Lorsqu'un coup décisif est possible pour le joueur au trait, à n'importe quel moment de la simulation (que ce soit dans l'arbre ou dans la partie Monte-Carlo), ce coup est joué immédiatement.
- Lorsqu'un coup décisif sera accessible au joueur adverse au prochain tour, on joue nécessairement de manière à l'empêcher d'avoir ce coup décisif.

Cette modification est possible à implémenter sans surcoût en temps de calcul (sous réserve d'implémentation soignée) ; on obtient alors d'excellents résultats, présentés en Table 1.

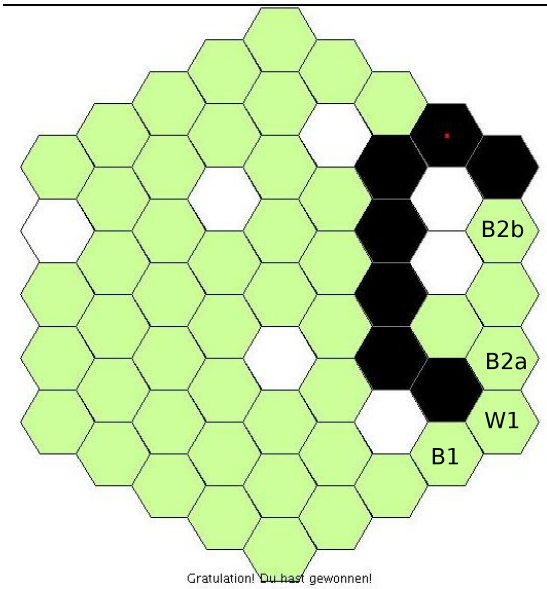
Conditions Expérimentales	Score contre la version sans coups décisifs
1000 simulations par coup	81.3 % ± 1.0%
20 secondes par coup	84.7 % ± 2.7%
30 secondes par coup	79.0 % ± 3.0%

Tab. 1 – Performance de la version avec coups décisifs contre la version sans coups décisifs.

6 Discussion

Nous pouvons clairement valider pour le jeu de Havannah, quelques techniques venant des programmes de jeu de Go, montrant la généralité de l'approche MCTS. Essentiellement :

- L'efficacité de la formule de Bernstein face à la formule de Hoeffding.
- Le taux de succès d'UCT avec $2k$ simulations par coup contre UCT avec k simulations par coup (presque 75% pour le jeu de Havannah, alors que c'est environ 63% pour le jeu de Go [11]). Ce taux semble indépendant de k.
- L'efficacité de l'heuristique RAVE est clairement validée. La principale force de RAVE est que l'avantage augmente avec la taille de l'espace d'actions, atteignant 100% de victoire, estimé sur 1347 parties en taille 8. D'un autre côté, RAVE devient moins efficace, et demande un paramétrage spécifique, lorsque le nombre de simulations par coup augmente - en



Taux de succès estimé par Havannah-Applet	Taux de succès estimé par notre programme
62.6%	46.4%
60.8%	47.2%
64.0%	51.2%
56.5%	54.8%
62.5%	65.4%
57.3%	63.7%
71.0%	88.8%
0.5%	

FIG. 2 – A gauche : résultat de la première partie jouée contre Havannah-Applet en taille 5. Notre programme gagne très rapidement, par une attaque multiple : l'adversaire blanc doit jouer *W1* (sans quoi noir gagne par pont). Alors noir peut connecter au côté en bas droit avec *B1*. Alors, noir a deux façons de se connecter au côté droit *B2a* et *B2b* ; blanc ne peut bloquer que l'une des deux et noir réalise alors une fourche. Á droite : taux de succès estimé pour chacun des joueurs ; on voit qu'Havannah-Applet ne suspectait pas cette attaque avant le dernier coup.

gardant toutefois un taux de succès de 66% dans les cas testés.

- Progressive widening, en dépit du fait qu'il a été montré dans [18] qu'il fait du sens même sans heuristique de classement des coups, n'est pas efficace dans notre cas. Dans le cadre du Go, il a été montré que c'était très efficace lorsque l'implémentation est basée sur des motifs locaux [10, 7], mais est néanmoins semble-t-il dépassé par des combinaisons linéaires entre score heuristique et score façon UCT[16].
- Notre programme peut battre Havannah-Applet facilement, alors que nous avons joué en tant que noir, et avec un temps par coup beaucoup plus faible. Faire d'autres expériences est difficile compte tenu du manque d'interface permettant de faire jouer automatiquement des programmes les uns contre les autres.
- La notion de coup décisif améliore grandement l'efficacité d'UCT (cette notion ne fait pas de sens au jeu de Go).

Remerciements. Nous remercions l'équipe MoGo, ainsi que Rémi Coulom, Tristan Cazenave, Bruno Bouzy, Victor Marsault, pour de nombreuses discussions intéressantes. Nous remercions aussi le projet ANR COSINUS (project EXPLO-RA nANR-08-COSI-004) pour son soutien.

Références

[1] J.-Y. Audibert, R. Munos, and C. Szepesvari. Use of variance estimation in the multi-armed bandit problem. In *NIPS 2006 Workshop on On-line Trading of Exploration and Exploitation*, 2006.

[2] P. Audouard, G. Chaslot, J.-B. Hoock, J. Perez, A. Rimmel, and O. Teytaud. Grid coevolution for adaptive simulations ; application to the building of opening books in the game of go. In *Proceedings of EvoGames*, 2009.

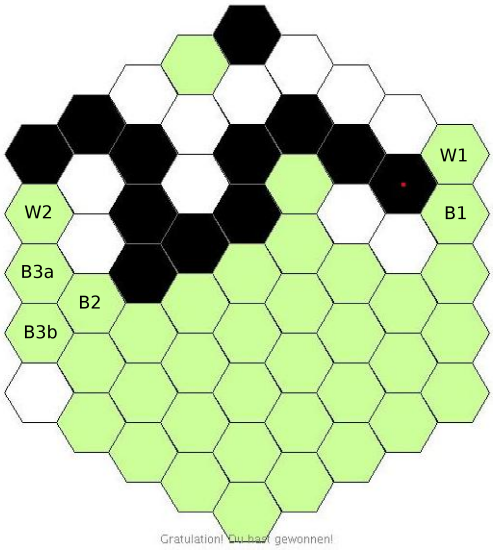
[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3) :235–256, 2002.

[4] B. Bruegmann. Monte carlo go. *Unpublished*, 1993.

[5] T. Cazenave and N. Jouandeau. On the parallelization of UCT. In *Proceedings of CGW07*, pages 93–101, 2007.

[6] G. Chaslot, J.-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik. Monte-Carlo Strategies for Computer Go. In P.-Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.

[7] G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy. Progressive strategies for monte-carlo tree search. In P. Wang et al., editors,



Taux de succès estimé par Havannah-Applet	Taux de succès estimé par notre programme
62.3%	45.7%
60.3%	48.2%
50.6%	49.9%
43.0%	55.9%
40.4%	52.6%
52.6%	55.4%
40.7%	56.2%
42.0%	63.5%
31.1%	60.0%
42.2%	75.8%
36.8%	68.6%
27.4%	78.1%
3.7%	

FIG. 3 – A gauche : résultat de la seconde partie jouée contre Havannah-Applet en taille 5. Notre programme joue noir et gagne par forfait. Blanc doit jouer *W1*, sans quoi noir réalise un pont. Alors, noir joue *B1* et se connecte à un deuxième côté. Ensuite blanc doit jouer *W2* (si noir joue *W2* alors noir a une fourche). Finalement, noir peut jouer *B2* et avec ce coup, il se connecte au troisième côté (par *B3a* ou *B3b* - blanc ne peut éviter ces deux possibilités). A droite : taux de succès estimé pour chacun des joueurs.

Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007), pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.

[8] G. Chaslot, M. Winands, and H. van den Herik. Parallel Monte-Carlo Tree Search. In *Proceedings of the Conference on Computers and Games 2008 (CG 2008)*, 2008.

[9] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.

[10] R. Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.

[11] S. Gelly, J. B. Hoock, A. Rimmel, O. Teytaud, and Y. Kalemkarian. The parallelization of monte-carlo planning. In *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO 2008)*, pages 198–203, 2008. To appear.

[12] S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *ICML '07 : Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press.

[13] H. Kato and I. Takeuchi. Parallel monte-carlo tree search with simulation servers. In *13th Game Programming Workshop (GPW-08)*, November 2008.

[14] L. Kocsis and C. Szepesvari. Bandit-based monte-carlo planning. In *ECML'06*, pages 282–293, 2006.

[15] T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6 :4–22, 1985.

[16] C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong. The computational intelligence of mogo revealed in taiwan's computer go tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, 2009 (accepted).

[17] V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical Bernstein stopping. In *ICML '08 : Proceedings of the 25th international conference on Machine learning*, pages 672–679, New York, NY, USA, 2008. ACM.

[18] Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, volume 21, 2008.

[19] Y. Wang and S. Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, pages 175–182, 2007.

[20] Wikipedia. Havannah, 2009.