# Utilisation de la recherche arborescente Monte-Carlo au Hex

**Tristan Cazenave*** — **Abdallah Saffidine****

* *Université Paris-Dauphine, LAMSADE*
*Place Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France*
*cazenave@lamsade.dauphine.fr*

** *LIASD - Université Paris 8*
*2 rue de la liberté, 93526 Saint-Denis Cedex*
*abdallah.saffidine@ens-lyon.fr*

RÉSUMÉ. *Nous présentons* YOPT, *un programme qui joue au Hex en utilisant des techniques de Monte-Carlo. Nous décrivons des heuristiques pour améliorer les simulations et les descentes d'arbre Monte-Carlo. Nous abordons aussi l'utilisation d'heuristiques pour améliorer la parallélisation du programme. Le niveau de* YOPT *atteint le niveau de* SIX *pour les temps utilisés en compétition.*

ABSTRACT. *We present* YOPT, *a program that plays Hex using Monte-Carlo tree search. We describe heuristics that improve simulations and tree search. We also address the use of heuristics that improve parallelization. The playing level of* YOPT *matches the playing level of* SIX *for playing times used in competition.*

MOTS-CLÉS : *recherche Monte-Carlo, Hex, RAVE, AMAF, parallélisation.*

KEYWORDS: *Monte-Carlo tree search, Hex, RAVE, AMAF, parallelization.*

## 1. Introduction

Le Hex est un jeu de plateau à deux joueurs, inventé indépendamment par le poète Piet Hein en 1942 et le prix Nobel d'économie John Nash en 1948. Il est le plus célèbre représentant de la catégorie des jeux de connexions. Bien qu'il soit proche des mathématiques par les théorèmes le concernant, le Hex a su développer une communauté de joueurs. Ainsi il y a quelques années était publié le premier livre de stratégie consacré au Hex (Browne, 2000).

Les règles du jeu sont élémentaires. Le plateau est un losange à pavage hexagonal dont chaque paire de bords opposés est balisée de la couleur d'un joueur. La figure 1 donne un exemple de plateau 6x6. Les compétitions de programmes se font actuellement sur des plateaux 11x11. Les joueurs posent chacun à leur tour un pion de leur couleur sur une case vide. Il n'y a ni prise ni déplacement et le gagnant est le joueur qui parvient à relier ses bords par une chaîne continue de pions. La figure 1 donne une position dans laquelle la partie est terminée.

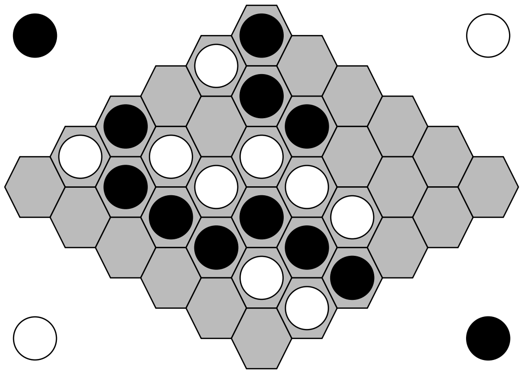

**Figure 1.** *Partie de Hex gagnée par Noir*

La démonstration de nombreuses propriétés sur le Hex est facilitée par la simplicité de ses règles. Ainsi, il est impossible d'aboutir à une partie nulle. D'une part, il n'existe pas de manière de remplir le plateau telle qu'une chaîne noire relie ses bords et une chaîne blanche relie les siens, d'autre part, si le plateau est saturé, alors l'un des joueurs possède une chaîne gagnante (Maarup, 2005). John Nash a montré qu'il existe une stratégie gagnante pour le joueur qui commence, mais la démonstration n'est pas constructive (voir A.2). Il est peut-être moins profitable au joueur non mathématicien d'apprendre que l'absence de partie nulle au Hex équivaut au théorème du point fixe de Brouwer en dimension deux (Gale, 1979). Il est possible de comprendre la difficulté de créer une intelligence artificielle de bon niveau au Hex par une étude de sa complexité : le problème de décision associé à la généralisation du Hex est PSPACE-complet (Even *et al.*, 1976; Reisch, 1981).

Lorsque l'on commence à jouer au Hex, deux remarques peuvent être utiles. La première est que le plateau se parcourt bien plus vite à l'aide de losanges (ou ponts), ces connexions sont pourtant aussi résistantes que la proximité parfaite. La seconde est que la force de sa position est celle de la zone la moins sûre, c'est en effet ici que l'adversaire a le plus de chance de réussir à franchir. Il semblerait qu'un débutant joue mieux en général lorsqu'il essaie d'empêcher son adversaire de gagner, plutôt que lorsqu'il se concentre sur son gain propre (bien que de par l'absence de parties nulles ces deux buts soient équivalents).

### 1.1. *Formes*

Dès lors que l'on repère un losange sur le plateau, on relie mentalement les groupes qu'il joint. Mémoriser cette forme permet d'éviter le calcul certes rapide de la connexion entre les deux groupes. Il possible de généraliser la notion de forme (en anglais *template*), certaines assurent par exemple la liaison entre un pion et le bord du plateau. La figure 2 donne un exemple de forme, nommée Ziggurate, qui relie une case au bord.

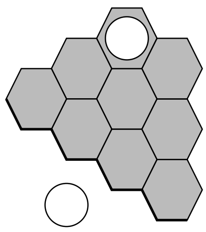

**Figure 2.** *La forme* Ziggurate *assure le lien entre le pion Blanc et le bord*

La méthode la plus simple pour prouver qu'une forme relie effectivement ses groupes est d'expliciter les menaces de connexions, chaque menace de connexion possédant un support. Le support (carrier) est l'ensemble des cases vides dont l'occupation affecte la réalisation de la connexion. Ainsi pour empêcher la connexion de se faire, l'adversaire doit jouer dans le support de la menace afin de la parer. Si les menaces sont nombreuses, il lui faut alors jouer dans l'intersection des supports. Si l'intersection est vide alors la connexion est prouvée, sinon il est possible d'étudier de manière plus exhaustive les quelques cas restants (voir figure 5). Un losange connecté au bord est un template de niveau deux puisque la distance au bord est de deux. La Ziggurate est un template de niveau trois qui se déduit de menaces de connexions comportant des losanges. Il existe aussi des templates de niveaux plus élevés. Ainsi la figure 3 donne un exemple de template de niveau quatre qui peut se déduire à l'aide de menaces de connexions impliquant des losanges et des Ziggurates (voir figure 4).

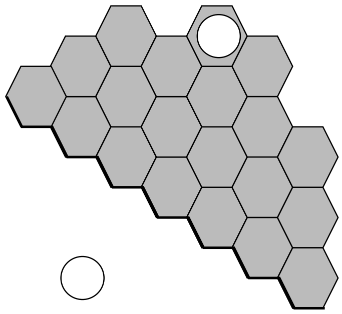

**Figure 3.** *La forme à distance IV*

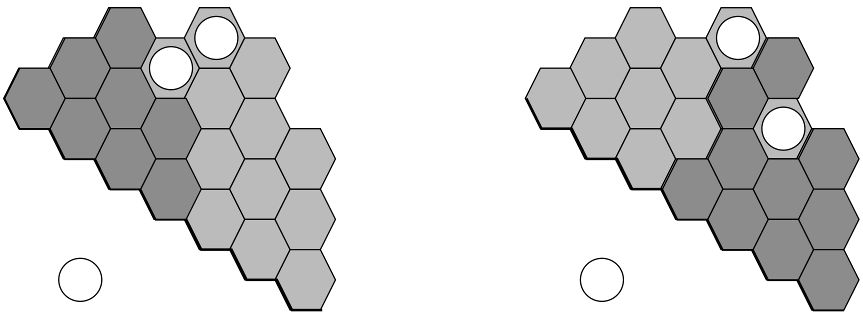

**Figure 4.** *Deux menaces blanches et leur support*

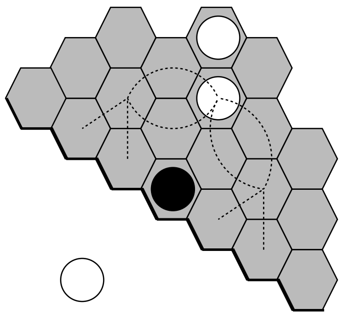

**Figure 5.** *Analyse d'une réponse de Noir dans la case restante*

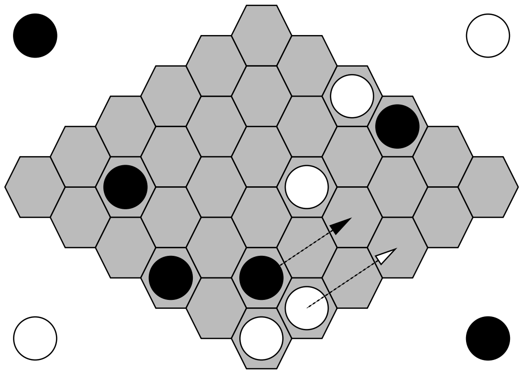

**Figure 6.** *Échelle imposée par Blanc pour laquelle Noir dispose d'une sortie. Noir au trait.*

## 1.2. *Échelles*

Les échelles, connues au Go également, imposent au joueur de Hex de considérer l'ensemble du plateau lors de son évaluation. Une échelle est la construction par l'adversaire d'un mur parallèle au bord vers lequel le groupe tend. La seule manière d'éviter d'être complètement bloqué consiste à disposer des sorties d'échelle (*ladder escape*) sur le chemin de l'échelle. Ces pions permettent alors de gagner l'initiative et de contourner le mur adverse (voir figure 6).

## 1.3. *L'intelligence artificielle au Hex*

Paradoxalement la fonction d'évaluation qui a donné les meilleurs résultats jusqu'à présent était fondée sur une idée de 1953 (Anshelevich, 2002), qui consiste à faire une analogie entre le plateau et un circuit électrique, et mesurer la résistance électrique entre les bords opposés. SIX, le programme qui a gagné les olympiades au Hex en 2003, 2004 et 2006 utilise une approche par connexions virtuelles similaire à celle de V. Anshelevich.

Avec notre programme YOPT nous avons choisi de tester une approche radicalement différente en utilisant des méthodes de recherche arborescente Monte-Carlo. Nos premiers essais d'utilisation des méthodes de Monte-Carlo au Hex remontent à l'année 2000. Toutefois à cette époque nous avions utilisé un Monte-Carlo sans recherche arborescente et les approches par connexions virtuelles du type de celles de V. Anshelevich donnaient de meilleurs résultats. Le succès récent des méthodes de développement d'arbres associées aux simulations Monte-Carlo au Go (Coulom, 2006; Kocsis *et al.*, 2006; Gelly *et al.*, 2006; Gelly *et al.*, 2007) ont donné un regain d'intérêt pour l'application de ces méthodes au Hex, intérêt qui est partagé par d'autres chercheurs dans ce domaine comme Rémi Coulom à l'université Lille 3 ou Philip Henderson à l'université d'Alberta. C'est pourquoi nous avons développé un programme de recherche Monte-Carlo pour Hex. L'IA réalisée, nommée *Yopt*, utilise l'algorithme UCT pour créer l'arbre des coups, et effectue des simulations Monte-Carlo aux feuilles. Ces simulations, toutefois, prennent en compte certaines spécificités du Hex.

Le reste de l'article est organisé de la façon suivante : la deuxième section traite de la recherche Monte-Carlo, la troisième section expose son application au Hex, la quatrième section donne les résultats expérimentaux.

## 2. Recherche Monte-Carlo

Nous abordons dans cette section les améliorations récentes des méthodes de Monte-Carlo appliquées aux jeux de réflexion. Nous nous intéressons particulièrement aux algorithmes UCT (Upper Confidence bounds for Trees) et RAVE (Rapid Action Value Estimation).

### 2.1. *Les simulations*

L'application des méthodes de Monte-Carlo aux jeux de réflexion fait appel à de nombreuses simulations aléatoires. Une simulation consiste à jouer une partie aléatoirement.

Pour améliorer les simulations on peut utiliser des connaissances qui biaisent les choix aléatoires vers les bons coups (Coulom, 2006; Gelly *et al.*, 2006; Cazenave, 2007). Au Go, par exemple, on interdit de jouer les coups qui bouchent les yeux dans les simulations. Ainsi, une partie se termine lorsqu'il n'y a plus que des chaînes vivantes et des yeux sur le goban. On peut alors précisément évaluer la position finale à l'aide des règles chinoises (le score d'un joueur est alors égal à la somme de ses pierres posées et de ses yeux). L'utilisation de connaissances dans les simulations n'est toutefois pas toujours bénéfique. Ainsi, dans notre programme de Go, il est arrivé qu'un joueur A de simulations aléatoires gagne la plupart du temps ses simulations contre un joueur B de simulations aléatoires car le joueur A utilisait plus de connaissances. Cependant une fois intégré dans un algorithme Monte-Carlo, l'algorithme Monte-Carlo utilisant les simulations aléatoires de B gagnait largement contre l'algorithme utilisant les simulations aléatoires de A. Il est clair que l'utilisation de connaissances bien choisies améliore un algorithme de Monte-Carlo, il faut toutefois être prudent sur les connaissances à utiliser dans les simulations car certaines connaissances que l'on pourrait juger à priori bénéfiques peuvent se révéler contre-productives.

L'algorithme 1 montre comment effectuer des simulations aléatoires au Hex. Les connaissances sur les coups urgents peuvent prendre la forme de suites de coups forcées, de captures dès que possible, ou tout autre schéma qui permettra à la simulation de garder une certaine rapidité. D'une manière générale il ne faut sélectionner comme coups urgents que les coups qui ont une très forte probabilité d'être les meilleurs, sinon les connaissances font baisser le niveau du programme. Une explication serait que certains aspects ou pièges ne soient pas repérés par les connaissances introduites, mais le soient parfois par la variété issue des tirages aléatoires.

**Algorithm 1** Lancé MC biaisé

```
recevoir une position
tant que la partie n'est pas finie faire
  si il y a un coup urgent alors
    jouer ce coup urgent
  sinon
    jouer un coup légal aléatoire
  fin de si
  changer le joueur courant
fin de tant que
renvoyer le gagnant
```

### 2.2. *Upper Confidence Bound*

L'application des méthodes Monte-Carlo aux jeux de réflexion fait appel à de nombreuses simulations aléatoires. L'exploitation standard de ces simulations se fait maintenant par l'algorithme UCT (Kocsis *et al.*, 2006), plus adapté que le min-max. Cet algorithme est apparu dans le cadre des problèmes de bandit-manchots où la seule manière d'obtenir des informations sur un état consiste à faire des tirages aléatoires.

Ces problèmes inspirés des machines à sous du casino proposent différentes manettes à actionner, chacune offrant une rétribution aléatoire selon une loi qui lui est propre. Il s'agit de trouver un équilibre entre exploiter les bras pour lesquels la loi de distribution des rétributions semble généreuse ; et explorer les bras qui ne l'ont pas encore été suffisamment pour obtenir des informations plus fiables sur leur espérance. Mieux explorer certains bras peut permettre de confirmer que leur distribution n'est pas favorable au joueur.

Kocsis et Szepesvari proposent dans (Kocsis *et al.*, 2006) d'adapter l'algorithme *Upper Confidence Bound* aux problèmes dans lesquels apparaissent des arbres (voir algorithmes 2, 3, 4) ; les problèmes de recherche de chemin ou les jeux de plateau en sont des exemples concrets. Dans le cas des jeux de plateau, on considère chacune des positions possibles comme un bras, les différentes évaluations effectuées depuis la position sont les rétributions de la manette. Toutefois il est possible de prendre en compte la structure arborescente de ces jeux : faire profiter les positions mères des simulations effectuées sur leurs filles. On aboutit à l'algorithme *Upper Confidence bound for Trees*(UCT)

Il est montré dans (Kocsis *et al.*, 2006) que la probabilité de choisir le meilleur coup convergeait vers 1 lorsque le nombre de simulations tendait vers l'infini. La formule qui donne la valeur UCT d'un nœud, en fonction de la moyenne $\mu$ des scores obtenus et du nombre $s$ de simulations qui sont passées par ce nœud et du nombre $n$ de simulations effectuées dans le père du nœud est :

$$\mu + C\sqrt{\frac{\log n}{s}}$$

avec $C$ la *Constante UCT* choisie par l'utilisateur (voir tableau 3).

UCT choisit de développer au début de chaque simulation aléatoire les coups qui amènent à la position dont la valeur UCT est maximale. La constante C permet de régler la degré d'exploration de l'algorithme. Elle dépend du domaine d'application. Plus la constante est grande plus l'algorithme aura tendance à explorer les coups moins bien notés.

L'algorithme UCT donne de très bons résultats au jeu de Go et les meilleurs programme de Go actuels comme MOGO (Gelly *et al.*, 2006; Gelly *et al.*, 2007) et CRAZY STONE (Coulom, 2006; Coulom, 2007) l'ont utilisé. Toutefois les versions récentes de MOGO et CRAZY STONE n'utilisent plus le terme après la moyenne, ils utilisent une constante UCT égale à 0. Nous verrons dans la partie résultats expérimentaux que c'est aussi le meilleur choix au Hex.

**Algorithm 2** Pseudo-code pour UCT

```
recevoir une position p
Soit A un arbre UCT vide à la racine près
pour Le nombre de descentes dans l'arbre choisi faire
   Soit p′ une copie de p
   Soit N le nœud résultat d'une descente dans l'arbre (A , p′)
   Soit R l'évaluation de p′
   On effectue une remontée de l'arbre (A , N , R)
fin de pour
renvoyer le coup fils de la racine de A qui a la meilleure valeur UCT
```

**Algorithm 3** Descente dans l'arbre

```
recevoir un arbre UCT,une position p
Soit N la racine de l'arbre
tant que Tous les fils de N ont déjà été exploré au moins une fois faire
   Soit F le fils de N ayant la plus grande valeur UCT
   On joue sur p le coup qui mène de N à F
   N ← F
fin de tant que
Soit F un fils de N tiré au hasard parmi les fils non exploré
F devient exploré
On joue sur p le coup qui mène de N à F
N ← F {N est maintenant une feuille}
renvoyer le nœud N
```

**Algorithm 4** Remontée dans l'arbre

**recevoir** un arbre UCT, un nœud $N$, le résultat $R$ d'une évaluation de $N$ {On suppose que pour l'évaluation d'une position équilibrée est 0}
**tant que** $N$ n'est pas la racine de l'arbre **faire**
  On incrémente le compteur de parties de $N$
  **si** $R$ a été réalisé alors que c'était le tour du joueur dont c'est le tour en $N$ **alors**
    On incrémente la valeur cumulée de $N$ de $R$
  **sinon**
    On incrémente la valeur cumulée de $N$ de $-R$
  **fin de si**
  $N \leftarrow$ le père de $N$
**fin de tant que**

### 2.3. *Monte-Carlo*

Pour profiter au mieux de l'algorithme UCT, les évaluations des feuilles doivent être les plus rapides possibles. En effet, dans la mesure où les rétributions des filles sont des rétributions pour la mère, il est profitable d'explorer l'arbre le plus en profondeur possible, le nœud racine recevant dans tous les cas la même quantité de rétribution.

Pour obtenir des évaluations rapides, le plus simple reste l'utilisation de simulations Monte-Carlo (MC) (voir algorithme 1). Elles consistent à simuler une continuation possible de la partie et renvoyer le résultat comme valeur de la position. La justification tient au fait qu'une position presque gagnée donnera lieu plus souvent à une victoire aléatoire qu'une position défavorable.

### 2.4. *RAVE*

L'algorithme RAVE (Rapid Action Value Estimation) (Gelly *et al.*, 2007) consiste à calculer à chaque nœud de l'arbre UCT les moyennes des coups qui ont été joués à n'importe quel moment dans les simulations aléatoires qui suivent le nœud.

L'heuristique qui consiste à calculer la moyenne des simulations dans lesquelles un coup a été joué à n'importe quel moment de la simulation, était déjà présente dans GOBBLE le premier programme de Monte-Carlo Go (Bruegmann, 1993). Cette heuristique est habituellement nommée AMAF (All Moves As First). L'apport de Sylvain Gelly a été de calculer AMAF à chaque nœud de l'arbre UCT et de la combiner efficacement avec la valeur UCT. En effet lorsque le nombre de simulations d'un nœud est faible, la moyenne des simulations du nœud est une mesure imprécise de l'intérêt du nœud alors que la valeur AMAF du coup qui mène à ce nœud est calculée sur beaucoup plus de simulations et a donc une variance plus faible.

Pour combiner AMAF avec UCT on utilise une constante k et un paramètre $\beta$ qui décroît de 1 à 0 ce qui permet de passer progressivement de la valeur AMAF à la valeur UCT. $\beta$ est calculé avec la formule suivante :

$$\beta = \sqrt{\frac{k}{3 \times s + k}}$$

On évalue alors l'intérêt d'un coup avec la formule suivante :

$$\beta \times AMAF + (1.0 - \beta) \times UCT$$

RAVE utilise cette formule pour choisir le coup à explorer lors de la descente de l'arbre au début de chaque simulation.

La formule originale de RAVE, que nous venons de présenter, a été améliorée par David Silver (Silver, 2008) et c'est cette dernière qui est utilisée dans les meilleurs programmes de Go.

## 3. Application au Hex

Nous commençons par exposer dans cette section les particularités des simulations Monte-Carlo au Hex, puis nous évoquons le développement de l'arbre de recherche ainsi que la parallélisation de l'algorithme.

### 3.1. *Les simulations*

L'utilisation de techniques de Monte-Carlo au Hex fait appel à des simulations aléatoires. La façon la plus simple de faire des simulations aléatoires au Hex consiste à choisir successivement pour chaque joueur une case vide au hasard et à la remplir. On remplit de cette façon le damier tant qu'il reste des cases vides. Une fois le damier rempli, on teste pour un des deux joueurs si ses deux bords sont connectés. Si c'est le cas, il a gagné la simulation, dans le cas contraire son adversaire a gagné.

Une amélioration des simulations complètement aléatoires consiste à répondre automatiquement aux menaces de déconnexion des losanges. Deux cases de la même couleur sont virtuellement connectées par un losange si elles ont deux voisines vides communes qui sont elles mêmes voisines. Ces deux cases sont virtuellement connectées puisque si l'adversaire joue sur une des deux voisines vides, le joueur peut alors remplir l'autre voisine vide et ainsi connecter les deux cases de sa couleur. Dans les simulations aléatoires, à chaque fois qu'un coup menace de déconnecter un losange en jouant sur une des deux cases vides voisines, le coup suivant est toujours de connecter en jouant l'autre case vide.

Au Hex, il existe des formes plus étendues que les losanges qui assurent des connexions virtuelles entre cases. Certaines formes que l'on rencontre souvent connectent une case au bord et sont appelées templates (cf figures 2 et 3). On peut détecter ces formes dans les simulations aléatoires et automatiquement répondre aux tentatives de déconnexion par les coups de connexion appropriés.

Une modification des simulations aléatoires consiste à mettre à jour à chaque coup les chaînes de cases connectées. On peut ainsi détecter la fin de partie avant que le damier ne soit complètement rempli.

Une particularité du Hex est qu'un coup d'un joueur ne détériore jamais la position pour ce joueur. Un coup est au pire inutile. On peut mettre en contraste cette propriété du Hex avec le Go. Au Go, il est nécessaire de détecter des mauvais coups pour que les simulations se terminent : un joueur pseudo-aléatoire ne doit pas se boucher un œil car c'est dans l'immense majorité des cas un mauvais coup. Cependant, contrairement au Go, il n'y a pas de capture au Hex et les simulations aléatoires se terminent donc naturellement sans nécessairement utiliser de connaissances.

### 3.2. *Condition de fin de partie*

Il est possible d'arrêter la simulation MC à plusieurs moments. On peut attendre que le plateau soit rempli, et profiter en ce sens de l'absence de partie nulle (voir A.1.2) qui garantit que l'évaluation de la partie ne changera pas par rapport à l'arrêt lorsque la partie est finie. Une troisième option est de cesser le jeu dès que l'un des joueurs est parvenu à relier ses bords par des formes (voir figure 7), en particulier des losanges. Dans ce cas non plus le résultat de la partie ne varie pas puisque les losanges sont automatiquement défendus.

La distinction entre les trois possibilités se fait lors de la propagation du résultat par RAVE. Par exemple on peut croire que continuer à jouer une fois que la partie est finie risque de bruiter la propagation par RAVE en récompensant des coups qui n'ont pas eu d'influence sur la partie. Les résultats expérimentaux ne confirment pas ce jugement. Ainsi le seul apport concret semble être le temps de réflexion qui diminue dans les positions presque gagnées.

### 3.3. *L'arbre de recherche*

L'algorithme RAVE s'applique naturellement au Hex et c'est l'algorithme que nous avons utilisé pour développer l'arbre au début de chaque simulation.

L'heuristique AMAF s'applique elle aussi très bien, puisqu'il n'y a pas de capture au Hex et que la structure locale d'une position est même plus stable au Hex qu'au Go.

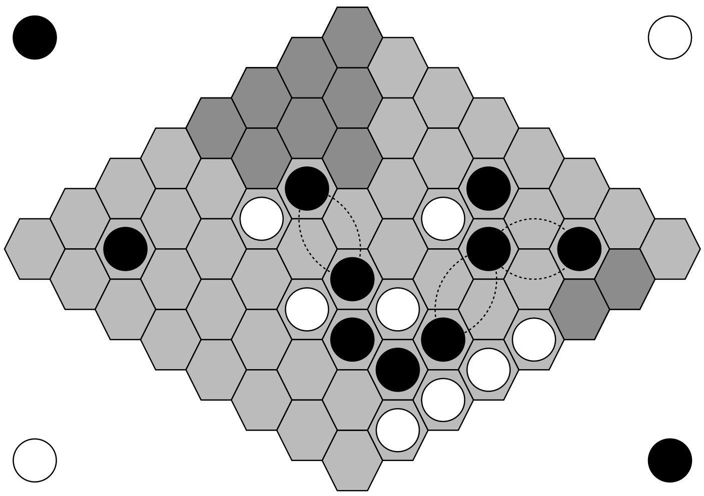

**Figure 7.** *Partie de Hex gagnée par Noir*

### 3.4. *La parallélisation*

La parallélisation d'UCT fait l'objet de recherches actives depuis son apparition (Cazenave *et al.*, 2007; Gelly *et al.*, 2008; Cazenave *et al.*, 2008; Chaslot *et al.*, 2008; Kato *et al.*, 2008). Il a été observé que le gain en rapidité qui peut être obtenu est dépendant de la rapidité des simulations (Cazenave *et al.*, 2008). Plus les simulations sont lentes, plus la parallélisation sera efficace. Une façon naturelle de ralentir les simulations sans trop perdre en niveau de jeu est d'effectuer plusieurs simulations par feuille de l'arbre UCT. C'est pourquoi nous avons testé YOPT avec différents nombres de simulations par feuille.

### 3.5. *Cases mortes*

Il existe dans certaines positions des cases dites *mortes* (voir figure 8) (Bjornsson *et al.*, 2007). L'issue de la partie ne dépend pas de l'occupation de ces cases. Il est donc possible de ne pas les évaluer dans la construction de l'arbre UCT, et d'empêcher les simulations aléatoires de jouer sur ces cases. Il est possible d'étendre ce raisonnement aux cases *capturées* (Bjornsson *et al.*, 2007), une case capturée par un joueur $J$ est une case telle que si l'adversaire de $J$ joue cette case alors il y a un coup pour $J$ qui fait de cette case une case morte (voir figure 9).

La détection des cases capturées et des cases mortes peut se faire efficacement en utilisant une bibliothèque de formes de petite taille et en les combinant pour en obtenir de nouvelles de taille plus grande. Cette détection n'a toutefois pas été implémentée.

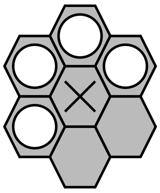

**Figure 8.** *La case marquée d'un X est* morte

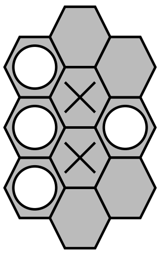

**Figure 9.** *Les cases marquées d'un X sont* capturées

## 4. Résultats expérimentaux

Il est nécessaire d'effectuer des séries de tests pour valider l'application de UCT au Hex, et pour comprendre l'influence des paramètres variables ainsi que celle des biais donnés aux simulations Monte-Carlo.

### 4.1. *Protocole*

Pour pouvoir juger efficacement de l'évolution du niveau entre plus de deux configurations, le plus simple est de les faire jouer contre un adversaire immuable de niveau semblable. Pour atteindre le niveau de SIX il fallait un grand nombre de simulations par coup (de l'ordre de 500 000), il n'était donc pas possible d'obtenir des statistiques fiables suffisamment rapidement. On conjecture que l'influence des paramètres sur la configuration lente est (très) similaire à celle sur la configuration rapide. Il a donc été choisi de fixer une configuration rapide (qui joue un coup en approximativement deux secondes) de YOPT, et de faire jouer contre cette version standard les configurations à tester. Les paramètres de l'IA standard sont donnés dans le tableau 1.

Le protocole est le suivant, 200 parties sont jouées sur un plateau de taille 11x11 contre l'IA standard, 100 avec les blancs et 100 avec les noirs, la partie commence

| Paramètre | Standard |
|---|---|
| Descentes | 16 000 |
| Simulations aux feuilles | 1 |
| Formes prises en compte | type3 |
| Constante RAVE | 16 000 |
| Constante UCT | 0,3 |

**Tableau 1.** *Paramètres de l'IA standard.*

| Descentes | 1 000 | 2 000 | 4 000 | 8 000 | 32 000 | 64 000 |
|---|---|---|---|---|---|---|
| Pourcentage de gain | 6 % | 11,5 % | 20 % | 33 % | 61 % | 68,5 % |

**Tableau 2.** *Variation du nombre de descentes.*

avec le premier coup noir a3. Cette ouverture permet un début de partie équilibré en contrebalançant l'avantage du trait. En général seul un paramètre est différent des paramètres standards.

### 4.2. *Paramètres variant dans la construction de l'arbre UCT*

Les paramètres suivants permettent d'ajuster la forme de l'arbre généré par UCT.

#### 4.2.1. *Nombre de descentes*

Ce paramètre correspond au nombre de descentes dans l'arbre UCT, la descente se concluant par une ou plusieurs simulations pseudo-aléatoires. Faire varier ce paramètre influence de manière notable (quasi-linéaire) le temps de réflexion. Le niveau, toutefois, bénéficie clairement d'un nombre de descentes plus élevé.

#### 4.2.2. *Constante UCT*

Cette constante permet d'ajuster le compromis exploration/exploitation de l'algorithme UCT. Lorsqu'elle est proche de 0, l'essentiel des calculs se fait pour distinguer les coups qui paraissent les plus prometteurs à première vue. Lorsqu'elle est plus grande, l'algorithme explore plus souvent les branches à priori moins prometteuses. Bien que l'influence sur le temps d'exécution soit négligeable, le niveau évolue de manière notable avec cette valeur. Lorsque UCT est utilisé sans RAVE, la meilleure constante est proche de 0,3. Cependant une fois que RAVE est utilisé on voit dans le tableau 3 que la meilleure constante est alors 0.

| Cste UCT | $-0,1$ | $0$ | $0,1$ | $0,2$ | $0,4$ | $0,5$ | $0,6$ | $0,7$ |
|---|---|---|---|---|---|---|---|---|
| % de gain | 38,5 % | 61 % | 60 % | 55,5 % | 42 % | 41 % | 35,5 % | 32,5 % |

**Tableau 3.** *Variation de la constante UCT*

| Constante RAVE | 0 | 8 000 | 32 000 | 64 000 | $\infty$ |
|---|---|---|---|---|---|
| Pourcentage de gain | 20 % | 40,5 % | 53 % | 52,5 % | 34 % |

**Tableau 4.** *Variation de la constante RAVE*

#### 4.2.3. *Constante RAVE*

L'heuristique AMAF est très bien adaptée au Hex où tous les coups permutent. Il est alors naturel de tester l'algorithme RAVE. On voit dans le tableau 4 que l'algorithme RAVE améliore sensiblement le niveau de YOPT. La valeur 0 qui correspond à la non-utilisation de RAVE ne gagne que 20 % de ses parties contre l'IA standard qui utilise une constante RAVE de 16 000. Les meilleures valeurs pour la constante sont assez élevées. Ainsi une constante RAVE infinie qui consiste à ne plus prendre en compte la moyenne mais seulement l'heuristique AMAF donne de meilleurs résultats que UCT (34 % contre 20 %).

### **4.3.** ***Paramètres variant dans l'évaluation des feuilles***

Lorsque l'on aboutit à une feuille de l'arbre UCT, on effectue une ou plusieurs simulations Monte-Carlo. Toutefois il est possible de forcer des suites de coups au sein de ces simulations. Ainsi, si l'on force la défense des losanges, un milieu de partie dans lequel un joueur a relié ses bords par des losanges sera évalué comme gagné par Monte-Carlo. De la sorte, la valeur d'une feuille est plus souvent correctement évaluée.

#### 4.3.1. *Simulations aux feuilles*

Il est possible d'utiliser la moyenne de plusieurs simulations Monte-Carlo comme valeur pour une feuille. Le temps de réflexion augmente quasi linéairement avec le nombre de simulations aux feuilles. Le niveau augmente sensiblement comme on le voit dans le tableau 5. La parallélisation de la recherche Monte-Carlo en devient d'autant plus intéressante.

#### 4.3.2. *Prise en compte des formes*

Il est possible d'influencer les simulations par l'ajout de réponses automatiques. On peut associer à toute intrusion dans une forme une réponse qui préserve la con-

| Nombre de simulations aux feuilles | 2 | 4 | 8 | 16 |
|---|---|---|---|---|
| Pourcentage de gain | 56 % | 67,5 % | 77,5 % | 74,5 % |

**Tableau 5.** *Variation du nombre de simulations aux feuilles*

| Formes prises en compte | type 1 | type 2 | type 4 |
|---|---|---|---|
| Pourcentage de gain | 22 % | 42 % | 71,5 % |

**Tableau 6.** *Variation de la prise en compte des formes*

nexion. Cela impose de passer plus de temps dans les simulations, mais l'amélioration de la qualité de jeu semble incontournable. Il est possible par ailleurs de choisir la quantité de formes à prendre en compte. Des tests ont été réalisés :

– type 1 : sans forme

– type 2 : avec les losanges seulement

– type 3 : avec les losanges et les formes de bord distance II

– type 4 : avec les losanges, les formes de bord distance II et les Ziggurates.

On voit dans le tableau 6 tout l'intérêt de l'utilisation des formes dans les simulations. Le temps supplémentaire passé à vérifier les formes dans les simulations n'est pas prohibitif et peut être contrebalancé par une meilleure parallélisation due au ralentissement des simulations. Des tests préliminaires à temps constant indiquent que la vérification de la forme à distance IV dans les simulations diminue le niveau de jeu. Tandis que les formes de type 4 augmente le niveau de jeu à temps constant également.

### 4.4. *Olympiades*

YOPT a participé aux Olympiades 2008 de l'International Computer Games Association (ICGA) à Pékin du 28 septembre au 5 octobre. Dans cette compétition de nombreux jeux de plateau abstraits sont représentés. La cadence est de 30 minutes par joueur par partie. Outre YOPT, se sont inscrits dans la section Hex en 2008 SIX, WOLVE et MOHEX. MOHEX utilisait aussi l'algorithme UCT avec en plus un solveur de positions appelé avant chaque coup pour détecter les gains par connexions virtuelles. WOLVE ainsi que SIX sont fondés sur les connexions virtuelles (Anshelevich, 2002) et diffèrent par la reconnaissance des cases mortes, et les choix dans le compromis entre recherches des connexions virtuelles et exploration de l'arbre par min-max.

YOPT a terminé quatrième. Il a toutefois battu SIX, le vainqueur des précédentes éditions, dans deux parties sur quatre et MOHEX dans une partie sur quatre. Il est prometteur qu'une approche Monte-Carlo aussi simple que celle de YOPT ait des résultats qui s'approchent du niveau des meilleurs programmes.

## 5. Conclusion

Nous avons présenté YOPT un programme de Hex qui utilise une approche Monte-Carlo. Son niveau atteint celui des meilleurs programmes comme SIX lorsqu'il effectue suffisamment de simulations, en utilisant toutefois des temps acceptables pour les compétitions entre programmes (une configuration avec 500 000 simulations par coups, soit une moyenne de 30 minutes par partie, permettait à YOPT de remporter plus de 40 % des parties contre SIX). Nous avons montré que le niveau de YOPT augmente avec le nombre de simulations, que l'heuristique RAVE donne de bons résultats au Hex, que l'utilisation de formes dans les simulations améliore le niveau de jeu, et que la parallélisation est prometteuse car l'augmentation du nombre de simulations par feuille augmente le niveau de jeu.

Les perspectives portent sur la reconnaissance des cases mortes, capturées et dominées, et sur l'utilisation des recherches $\lambda$ (Thomsen, 2000) ou des menaces généralisées (Cazenave, 2003) qui devrait permettre de pallier l'une des principales lacunes de YOPT : la mauvaise gestion des échelles (ce problème se pose aussi pour les méthodes de Monte-Carlo au Go). Finalement l'application des connexions virtuelles dans les simulations, au lieu de schémas de reconnaissance prédéfinis, pourrait les biaiser de manière plus pertinente.

Remerciements

Nous tenons à remercier Philip Henderson qui nous a informé que l'heuristique qui consiste à répondre automatiquement aux menaces de déconnexion des losanges dans les simulations améliorait son programme de Monte-Carlo Hex, ainsi que Rémi Coulom pour avoir renouvelé notre intérêt pour l'approche Monte-Carlo au Hex.

## 6. Bibliographie

Anshelevich V., « A hierarchical approach to computer Hex », *Artificial Intelligence*, vol. 134, n° 1-2, p. 101-120, 2002.

Bjornsson Y., Hayward R., Johanson M., van Rijswijck J., « Dead Cell Analysis in Hex and the Shannon Game », *Graph Theory in Paris : Proceedings of a Conference in Memory of Claude Berge (GT04 Paris)*, p. 45-60, 2007.

Browne C., *Hex Strategy. Making the Right Connections*, A K PETERS, 2000.

Bruegmann B., « Monte-Carlo Go », , Report, 1993. http ://citeseer.ist.psu.edu/637134.html.

Cazenave T., « A Generalized Threats Search Algorithm », *Computers and Games 2002*, vol. 2883 of *Lecture Notes in Computer Science*, Springer, Edmonton, Canada, p. 75-87, 2003.

Cazenave T., « Playing the Right Atari », *ICGA Journal*, vol. 30, n° 1, p. 35-42, March, 2007.

Cazenave T., Jouandeau N., « On the parallelization of UCT », *Computer Games Workshop 2007*, Amsterdam, The Netherlands, p. 93-101, June, 2007.

Cazenave T., Jouandeau N., « A Parallel Monte-Carlo Tree Search Algorithm », *Computers and Games*, LNCS, Springer, Beijing, China, p. 72-80, 2008.

Chaslot G., Winands M. H. M., van den Herik H. J., « Parallel Monte-Carlo Tree Search », *Computers and Games*, vol. 5131 of *Lecture Notes in Computer Science*, Springer, p. 60-71, 2008.

Coulom R., « Efficient Selectivity and Back-up Operators in Monte-Carlo Tree Search », *Computers and Games 2006*, Volume 4630 of LNCS, Springer, Torino, Italy, p. 72-83, 2006.

Coulom R., « Computing Elo Ratings of Move Patterns in the Game of Go », *ICGA Journal*, vol. 30, n° 4, p. 198-208, December, 2007.

Even S., Tarjan R. E., « A Combinatorial Problem Which is Complete in Polynomial Space », *Journal of the Association for Computing Machinery*, October, 1976.

Gale D., « The game of Hex and the Brouwer fixed-point theorem », *Amer. Math. Monthly*, vol. 86, n° 10, p. 818-827, 1979.

Gelly S., Hoock J.-B., Rimmel A., Teytaud O., Kalemkarian Y., « The Parallelization of Monte-Carlo Planning », *ICINCO*, 2008.

Gelly S., Silver D., « Combining online and offline knowledge in UCT », *ICML*, p. 273-280, 2007.

Gelly S., Wang Y., Munos R., Teytaud O., Modification of UCT with Patterns in Monte-Carlo Go, Technical Report n° 6062, INRIA, 2006.

Hexwiki, Website, 2008. http ://www.hexwiki.org.

Kato H., Takeuchi I., « Parallel Monte-Carlo Tree Search with Simulation Servers », *13th Game Programming Workshop (GPW-08)*, November, 2008.

Kocsis L., Szepesvàri C., « Bandit based monte-carlo planning », *ECML*, vol. 4212 of *Lecture Notes in Computer Science*, Springer, p. 282-293, 2006.

Maarup T., Hex Everything You Always Wanted to Know About Hex But Were Afraid to Ask, Master's thesis, University of Southern Denmark, 2005.

Reisch S., « Hex ist PSPACE-vollständig », *Acta Inf.*, vol. 15, p. 167-191, 1981.

Silver D., , 2008. http ://computer-go.org/pipermail/computer-go/2008-February/014095.html.

Thomsen T., « Lambda-search in game trees - with application to Go », *ICGA Journal*, vol. 23, n° 4, p. 203-217, 2000.

## A. Démonstrations de quelques propriétés

### A.1. *À propos des parties nulles*

#### A.1.1. *Le Y*

Le Y est une généralisation du Hex inventée en 1970 par Charles Titus et Craige Schensted. Les règles sont aussi simples qu'au Hex, on joue sur un plateau triangulaire à pavage hexagonal, le gagnant est le premier joueur parvenant à relier les trois bords par une même chaîne. Le Y est plus symétrique que le Hex : il n'y a pas de notion

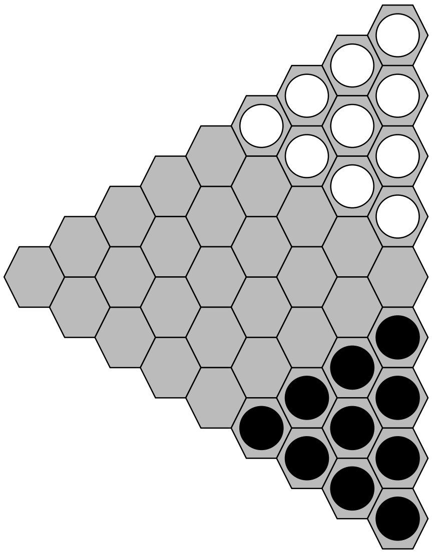

**Figure 10.** *Partie de Y prête à simuler une partie de Hex*

d'appartenance des bords. On y retrouve la plupart des notions de Hex : importance du centre, utilisation des formes... Un bon joueur de Hex est souvent un bon joueur de Y et vice et versa.

Toute partie de Hex peut être vue comme un cas particulier d'une partie de Y dans laquelle un certain nombre de coups préliminaires auraient été effectués (voir figure 10).

A.1.2. *Absence de partie nulle*

Il suffit de montrer que le Y n'admet pas de partie nulle pour avoir ce résultat au Hex. L'absence de partie nulle couvre en fait deux propriétés distinctes. Il n'y a jamais deux chaînes gagnantes sur un plateau de Y ; sur tout plateau complètement rempli il existe une chaîne gagnante.

A.1.2.1.

La première propriété ne pose pas de difficulté. Supposons une position de Y avec deux chaînes gagnantes. Chaque bord est en contact avec les deux autres et avec les deux chaînes, qui sont elles-mêmes en contacts avec les trois bords et l'une avec l'autre (cette dernière affirmation peut être obtenue par un processus de gonflement qui ne change rien au problème). On vient donc d'obtenir le graphe complet à 5 sommets. Or ce graphe n'est pas planaire, ce qui montre la propriété par l'absurde.

A.1.2.2.

Il n'est développé ici qu'une esquisse de la démonstration, pour la démonstration complète voir (Hexwiki, 2008). On définit le statut d'un plateau comme la valeur de vérité de la proposition “Il existe au moins une chaîne gagnante”. Soit un plateau $P$ rempli, on va montrer que le statut de $P$ est le même que le statut d'un autre plateau pour lequel on peut connaître le statut de manière triviale. Il est possible de classer les groupes de pions suivant le nombre de côtés qu'ils touchent. Le plateau $P\prime$ dans lequel tous les groupes ne touchant pas de bord sont remplacés par la couleur les encerclant a le même statut que $P$. De ce plateau $P\prime$, on peut remplacer tous les groupes ne touchant qu'un bord par la couleur les encerclant tout en conservant le statut. De même avec les groupes touchant deux bords. Le statut est toujours conservé, et dans chaque cas il reste toujours au moins un groupe (celui qui encerclait) sur le plateau. Dans le dernier plateau, tous les groupes touchent 3 bords, et il y a au moins un groupe. Ainsi sur le plateau $P$ il y avait déjà une chaîne gagnante.

### A.2. *Stratégie gagnante pour le premier*

Le Hex est un jeu déterministe et à information parfaite et complète, il y a un nombre fini de parties possibles. Il n'y a pas de partie nulle. Il est donc possible de créer l'arbre de jeu entier issu de la position initiale. Les feuilles sont notées suivant l'issue de la partie. Il s'agit ensuite de noter par récurrence tous les nœuds jusqu'à arriver à la racine de l'arbre. Du point de vue du joueur A, un nœud obtenu après un coup de A est gagnant si et seulement si tous ses fils sont gagnant (c'est-à-dire que tous les coups de B l'emmènent vers la défaite). Un nœud obtenu après un coup de B est gagnant (pour A) si et seulement si il existe un fils gagnant. Si la racine est notée comme défaite pour le joueur A, alors il existe une stratégie gagnante pour le joueur B. Sinon il en existe une pour le joueur A.

Montrons par l'absurde qu'il n'existe pas de stratégie gagnante pour le joueur B en début de partie. Pour cela remarquons d'abord que pour toute position où un joueur est gagnant, l'ajout de pions pour ce joueur ne change pas l'issue de la partie en cas de jeu parfait. Autrement dit il n'y a pas de zugzwang au Hex. Supposons maintenant que le joueur B ait une stratégie gagnante. Le joueur A commence par jouer un coup aléatoire, puis applique la stratégie gagnante en tant que deuxième joueur comme si son coup aléatoire n'avait pas été joué et que la case était encore vide. Dès que le joueur A doit jouer sur la case de son premier coup, il joue un autre coup aléatoire qu'il ignorera aussi bien. Ainsi les deux joueurs possèdent une stratégie gagnante ce qui n'est pas possible. Le joueur B n'a donc pas de stratégie gagnante. Comme les parties nulles sont impossible, alors il existe une stratégie gagnante pour le joueur A.

Cette démonstration est appelée vol de stratégie, et peut être appliquée à tout jeu à information complète et parfaite, ne présentant pas de hasard, pour lequel les rôles de A et B sont symétriques, et qui ne présente pas de zugzwang.

**ANNEXE POUR LE SERVICE FABRICATION**
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
   *RSTI - RIA – 23/2009. Jeux : modélisation et décision*
2. AUTEURS :
   *Tristan Cazenave* — Abdallah Saffidine***
3. TITRE DE L'ARTICLE :
   *Utilisation de la recherche arborescente Monte-Carlo au Hex*
4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   *Recherche Monte-Carlo au Hex*
5. DATE DE CETTE VERSION :
   *23 mars 2009*
6. COORDONNÉES DES AUTEURS :
   - adresse postale :
     * Université Paris-Dauphine, LAMSADE
     Place Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France
     cazenave@lamsade.dauphine.fr
     ** LIASD - Université Paris 8
     2 rue de la liberté, 93526 Saint-Denis Cedex
     abdallah.saffidine@ens-lyon.fr
   - téléphone : 00 00 00 00 00
   - télécopie : 00 00 00 00 00
   - e-mail : cazenave@lamsade.dauphine.fr
7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.23 du 17/11/2005.
8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com