Fonction d'évaluation

1 Introduction

Ce document présente quelques principes pour l'élaboration de fonctions d'évaluation pour le jeu d'Othello. La réalisation d'une fonction d'évaluation performante fait nécéssairement appel à l'inventivité et la créativité de chacun, ce pourquoi cette partie du projet est certainement la plus intéressante (et, éventuellement, la plus difficile) des différentes parties.

2 Fonctions d'évaluation simples

2.1 Précision et rapidité

On se rappelle que dans le cadre de la programmation d'un jeu de réflexion, la fonction d'évaluation est utilisée au niveau des feuilles de l'arbre de recherche (développé au cours de la recherche MinMax/AlphaBeta). Elle doit être capable d'estimer la qualité d'une position le plus précisément possible : la valeur calculée (généralement numérique, mais qui peut être également binaire dans le cas de jeux très simples, où chaque valeur correspond à victoire ou défaite – une troisième valeur pour un match nul peut également être utilisée) doit fournir une information la plus fiable possible sur l'issue de la partie la plus probable à laquelle cette position peut mener.

Le fait que la fonction d'évaluation soit appelée et utilisée au niveau des feuilles des arbres de rechecche et que quelques centaines de milliers de feuilles (au moins) seront développées à chaque appel de la fonction MinMax (ou AlphaBeta) souligne par ailleurs l'importance de la vitesse d'exécution de la fonction d'évaluation. S'il peut sembler « facile » de construire une fonction d'évaluation fiable (pour cela, on peut par exemple envisager de créer des milliers de **critères** – cf. ci-après – à combiner et de trouver une pondération adéquate entre ces milliers de critères – tâche qui n'est pas nécessairement simple – pour l'évaluation de la qualité d'une position), il faut toujours garder en tête que le temps d'exécution est un facteur limitant. Un compromis fiabilité/vitesse doit donc être trouvé pour la fonction d'évaluation.

2.2 Evaluation reposant sur des critères

Une stratégie assez naturelle pour la réalisation d'une fonction d'évaluation repose sur la définition et la combinaison de critères caractérisant une position. Cette stratégie donne lieu à des fonctions d'évaluation de la forme :

$$f(p) = \alpha_1 c_1(p) + \ldots + \alpha_m c_m(p)$$

où p est une position de jeu (c'est-à-dire, une configuration donnée du damier – **attention :** une position n'est donc pas simplement une case de l'othellier mais la disposition des pions sur l'ensemble des 64 cases de l'othellier à un moment donné), les α_i sont des réels et les c_i des critères permettant de mesurer une caractéristique de la position p.

Dans le cas d'une fonction d'évaluation pour Othello, ce type de fonction d'évaluation peut s'instancier de la manière suivante :

$$f(p) = \alpha_1 \text{mobilité}(p) + \alpha_2 \text{matériel}(p) + \alpha_3 \text{coins}(p)$$

où (par exemple) :

mobilité(p) est la différence de mobilité entre le joueur ayant le trait et le joueur adverse, pour la position p ;

matériel(p) est la différence entre le nombre de pions du joueur ayant le trait et le joueur adverse, pour la position p ;

coins(p) est la différence entre le nombre de coins possédés par le joueur ayant le trait et le joueur adverse, pour la position p.

Deux problèmes doivent être abordés pour l'utilisation de ce type de fonction d'évaluation :

1. la définition de critères ;
2. la choix des paramètres $\alpha_1, \ldots, \alpha_m$

A ces deux problèmes vient évidemment s'ajouter celui de la programmation efficace de fonctions permettant l'évaluation des critères (le calcul de la note finale donnée par la fonction d'évaluation consiste simplement en une combinaison linéaire de critères en fonction de $\alpha_1, \ldots, \alpha_m$; cette phase est donc très simple et ne nécessite pas d'optimisation particulière).

Concernant la définition de critères, c'est essentiellement l'expérience de jeu et/ou la lecture d'articles et de documents sur les stratégies importantes pour Othello qui permettent de définir des critères pertinents. Sans entrer dans les détails (car cette partie du travail doit faire appel à la créativité de chacun), quelques concepts à prendre en compte, et donc, à partir desquels il peut être pertinent de définir des critères sont : la mobilité, le matériel (i.e. le nombre de pions), le nombre de coins, la mobilité potentielle (ou encore le nombre de pions « frontière »), la centralité de la position, le nombre de pions définitifs, etc. Bien d'autre critères peuvent être envisagés.

Pour le choix des paramètres de la combinaison linéaire, une première stratégie est de fixer ces paramètres à la main. Par exemple, il peut être considéré que la possession de coins est quelque chose de primordial tout comme le fait d'avoir des pions définitifs (c'est-à-dire des pions que l'adversaire ne peut plus retourner) et que donc, des coefficients élevés doivent être associés à ces critères alors que des coefficients moins élevés peuvent être attribués aux critères tels que le matériel ou encore la centralité. Il est cependant bien évident qu'une méthode automatique de choix de ces coefficients est préférable, en particulier si cette méthode repose sur l'expérience tirée de parties jouées. La dernière partie de ce document, qui traite des algorithmes génétiques, propose une stratégie (qu'il faudra modifier pour l'adapter au cas de la fonction d'évaluation d'Othello) pour déterminer automatiquement les coefficients.

2.3 Découpage d'une partie en phases

Une remarque importante concernant la réalisation d'une fonction d'évaluation : tous les critères n'ont pas la même importance tout au long du déroulement de la partie et un même critère

peut avoir un coefficient positif au début de la partie puis négatif par la suite. En effet, si l'on prend l'exemple du nombre de pions possédés par un joueur, il est très important qu'au début de la partie ce nombre soit plutôt faible (et donc qu'un coefficient négatif soit affecté au critère matériel) alors qu'il doit évidemment être le plus élevé possible à la fin de la partie (le coefficient associé au matériel doit donc être positif).

Afin de prendre en compte cette remarque, une stratégie évidente est donc de disposer de plusieurs fonctions d'évaluation, c'est-à-dire plusieurs jeux de coefficients $\alpha_1, \ldots, \alpha_m$, chacune d'elle étant associée à une phase précise de la partie. La délimitation des phases de jeu peut se faire très facilement par rapport au nombre de pions présents sur le damier. Un découpage assez naturel est par exemple de considérer qu'une partie d'Othello se découpe en 3 phases : le début de partie, le milieu de partie et la fin de partie. Un découpage peut supposer que le début de partie est caractérisé par la présence de 0 à 20 pions sur le damier, le milieu la présence de 20 à 40 pions puis la fin de partie par la présence de 40 à 64 pions (ce découpage est donné à titre purement indicatif et n'est pas forcément optimal). En poussant plus loin cette idée de phases, les meilleurs programmes en utilisent une vingtaine : les plateaux de 4 à 6 pions, ceux de 7 à 9 pions, ceux de 10 à 12 pions, etc. Par conséquent, ces programmes maintiennent autant (c'est-à-dire une vingtaine) de jeux de coefficients $\alpha_1, \ldots, \alpha_m$ (un jeu de coefficients pour chaque phase). Si l'utilisation de plusieurs jeux de coefficients permet d'obtenir des programmes extrêmement forts, cette approche nécessite de façon critique d'avoir à disposition un mécanisme de détermination automatique des coefficients.

3 Algorithmes génétiques

Les algorithmes génétiques constituent une famille de méthodes d'optimisation globale. Ils sont particulièrement bien adaptés à la recherche de solutions de problèmes d'optimisation (i.e. des problèmes où l'on veut minimiser ou maximiser la valeur d'une fonction – dans le cas d'Othello, la fonction à optimiser est celle qui calcule la proportion de victoires et de défaites) où la fonction à optimiser peut ne pas être dérivable et possède plusieurs minima locaux (voir Figure 1, qui représente quelques situations pour une fonction à une variable).

3.1 Idée générale

La paradigme implémenté par les algorithmes génétique suit l'idée de Darwin qui dit que les individus d'une population les plus adaptés à un environnement sont ceux qui ont le plus de chances de se reproduire et donc de donner naissance à une nouvelle génération elle-même mieux adaptée que la génération précédente par le jeu de la transmission de caractéristiques via le code génétique.

L'idée se traduit informatiquement par la disposition de plusieurs éléments :

Une mesure de fitness. Cette mesure permet d'évaluer le degré d'adaptation de chaque individu à son environnement.

Un opérateur de sélection. Cet opérateur permet de sélectionner parmi tous les individus d'une population ceux qui sont les plus aptes à se reproduire. Cette sélection se fait

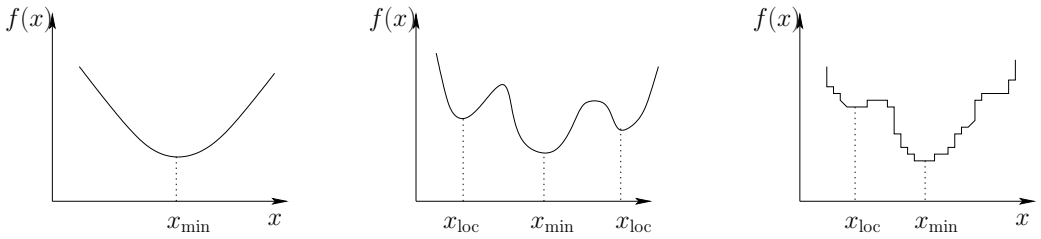


Figure 1: Minima locaux/globaux et fonction non dérivable. La figure de gauche représente le cas le plus favorable pour le problème d'optimisation : la fonction f est dérivable (la courbe est « douce » et possède un minimum global $x_{\min}$ (aucune autre valeur de x autre que $x_{\min}$ ne permet d'avoir $f(x) < f(x_{\min})$) ; pour ce genre de fonction, il existe plusieurs algorithmes permettant de trouver un minimum global. Dans le cas de la figure du milieu, la fonction est également dérivable mais présente cette fois plusieurs minima : deux minima locaux (c'est-à-dire qu'à proximité des valeurs x_{loc}, il n'y a pas de valeurs x permettant d'avoir $f(x) < f(x_{\text{loc}})$) et un minimum global $x_{\min}$; des méthodes pour identifier des minima locaux efficacement existent ($x_{\min}$ est un minimum global **et** local) mais dans le cas général, il est très dur de trouver un minimum global. La fonction de droite n'est pas dérivable et possède deux minima, un local et un global ; la recherche d'un minimum global dans ces conditions est assez difficile et les algorithmes génétiques peuvent se présenter comme une solution possible pour résoudre ce type de problème.

généralement par rapport à la fitness des individus et un individu peut être sélectionné plusieurs fois.

Un codage pour les génomes (les individus). La disposition d'un codage pour la représentation informatique d'un individu ou génome est nécessaire pour la réalisation de l'opération de reproduction.

Un opérateur de reproduction. Cet opérateur de reproduction permet de générer à partir de deux individus ou génomes, d'autres génomes (en général, deux) qui seront les enfants des deux individus se reproduisant. La reproduction doit permettre la transmission des génomes des deux parents aux enfants. Cet opérateur de reproduction fait généralement appel à deux procédures : une procédure de *croisement*, qui consiste en l'échange de portions de génomes entre les deux génomes parent et une procédure de *mutation* où certains gènes, dans les génomes enfant obtenus sont changés aléatoirement avec une très faible probabilité. Il faut également noter que le processus de croisement peut se contenter de recopier les génomes des parents : les enfants sont les clones des parents.

A partir de ces éléments, on obtient l'algorithme génétique de haut-niveau décrit par l'Algorithme 1. La section suivante instancie et illustre cet algorithme sur un problème d'optimisation assez simple.

3.2 Un exemple : le meilleur pickpocket

Dans cette section nous donnons un exemple d'algorithme génétique pour trouver le meilleur génome pour un pickpocket. Des opérateurs simples de sélection et de reproduction sont proposés ; ces opérateurs pourront servir de base à un algorithme génétique d'apprentissage des coefficients d'une fonction d'évaluation reposant sur l'utilisation de critères.

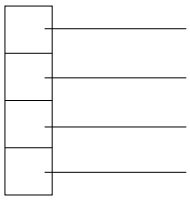
La question qui est posée dans cette section est celle de savoir quel est le meilleur génome pour

Algorithm 1 Algorithme génétique générique

Entrées : n, T
Sortie : $\mathbf{g}$, génome ayant la meilleure fitness à la génération T

Créér une population P_0 de n génomes aléatoires : $P_0 = \{\mathbf{g}_1^0, \ldots, \mathbf{g}_n^0\}$
for $t = 1, \ldots, T$ **do**
· calculer la fitness f_i de chaque génome $\mathbf{g}_i$
· sélectionner avec replacement (i.e., un génome peut être sélectionné plusieurs fois) n génomes dans P_t, ces individus sont $\mathbf{g}'_1, \ldots, \mathbf{g}'_n$
· chaque paire $(\mathbf{g}'_1, \mathbf{g}'_2)$, $(\mathbf{g}'_3, \mathbf{g}'_4), \ldots, (\mathbf{g}'_{n-1}, \mathbf{g}'_n)$ se reproduit pour donner les paires $(\mathbf{g}_1^t, \mathbf{g}_2^t)$, $(\mathbf{g}_3^t, \mathbf{g}_4^t), \ldots, (\mathbf{g}_{n-1}^t, \mathbf{g}_n^t)$ (ici, chaque paire donne naissance à deux paires)
· $P_t := \{\mathbf{g}_1^t, \ldots, \mathbf{g}_n^t\}$
end for
renvoyer l'individu ayant la meilleure fitness dans P_T

un pickpocket, c'est-à-dire celui qui permet à un pickpocket d'avoir le plus de bénéfices des ses activités (délictueuses, bien entendu). Supposons que le génome d'un pickpocket soit codé sous la forme d'un vecteur de 4 bits de la manière suivante :



où chaque entrée du vecteur (qui correspond à chaque gène du génome) vaut 1 si le voleur dispose effectivement du caractère correspondant et de 0 sinon. Il est évident que le meilleur génome est celui qui n'est composé que de 1. Nous allons voir comment un algorithme génétique peut réussir à trouver automatiquement cet individu. Pour cela, nous définissons tout d'abord les éléments nécessaires au fonctionnement de l'algorithme génétique.

Fitness. Ici, la fitness sera tout simplement la somme des bits dans le génome : plus un génome présente les caractéristiques favorables pour être un bon pickpocket plus le nombre de bits valant 1 est grand ; cette fonction d'évaluation est donc assez naturelle. Bien évidemment, dans les cas d'utilisation plus généraux d'algorithmes génétiques, les fonctions de fitness (i.e. les fonctions à optimiser) sont bien plus compliquées (voir plus loin pour l'utilisation d'algorithmes génétiques pour l'apprentissage d'une fonction d'évaluation pour le jeu d'Othello). On peut néanmoins se rendre compte dès à présent que cette fonction de fitness présente une caractéristique usuellement assez compliquée à prendre en compte qui est qu'elle n'est pas continue (et donc pas dérivable). Ce type de fonction est très difficile à optimiser en général.

Sélection. La sélection des génomes qui vont se reproduire se fait simplement de la manière suivante. Tout d'abord, on calcule les fitness f_i des n génomes de la population et on associe

à chacun d'entre eux la probabilité p_i d'être sélectionné calculée par :

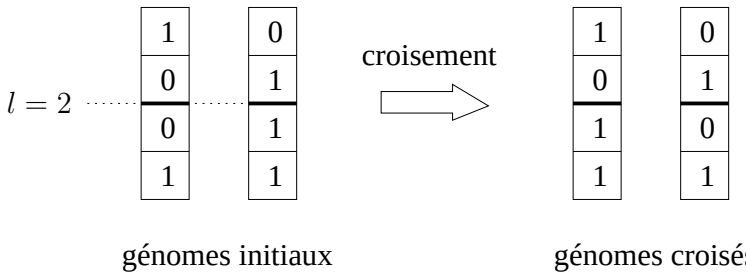
$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i}.$$

Un génome est alors sélectionné pour la reproduction en fonction de sa probabilité p_i (cf. la multitude d'informations sur la roue de loterie biaisée que donne Google).

Reproduction. Pour un couple de génomes donné, la reproduction se déroule en trois temps.

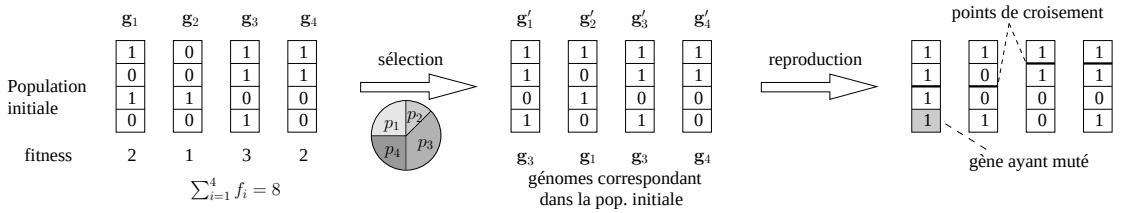
croisement ? Tout d'abord, il est décidé en fonction d'une probabilité de croisement p_c fixée au début de l'algorithme (cette probabilité peut prendre des valeurs entre 0.3 et 0.9) si un croisement des génomes du couple va effectivement être mis en œuvre (il suffit de tirer un nombre aléatoire entre 0 et 1 et si ce nombre est inférieur à p_c alors un croisement va avoir lieu, aucun croisement n'ayant lieu sinon). Si le couple ne doit pas se reproduire alors l'opération de mutation est effectuée, sinon l'opération de croisement en tant que telle est effectuée.

croisement Un nombre entier aléatoire l est choisi entre 1 et 3 (s'il y avait eu 5 bits, on aurait choisi un nombre entre 1 et 4, et plus généralement, dans le cas de m bits, on aurait choisi l entre 1 et $m-1$) ; l correspond à l'endroit où va avoir lieu le croisement entre les deux génomes étudiés. Par exemple, si $l = 2$ alors les deux génomes enfant sont créés de la manière suivante : le premier génome enfant est constitué des gènes 1 à 2 (c'est-à-dire l) du premier génome parent et des gènes 3 (c'est-à-dire $l+1$) à 4 du second génome parent, alors que le second génome enfant est constitué des gènes 1 à 2 (c'est-à-dire l) du second génome parent et des gènes 3 (c'est-à-dire $l+1$) à 4 du premier génome parent. La figure suivante illustre ce processus :



mutation Le processus de mutation choisi ici est très simple. Pour chaque génome considéré (qui peut être obtenu du croisement ou bien de la recopie de génomes, cf. deux points précédents), on choisit d'inverser chaque bit en fonction d'une probabilité de mutation p_m (généralement très petite : par exemple 0.05) ; plus concrètement, pour chaque bit de chaque génome, on génère un nombre aléatoire entre 0 et 1, si ce nombre aléatoire est plus petit que p_m alors le bit considéré est inversé et il reste inchangé sinon.

Le schéma suivant illustre une itération complète de l'algorithme génétique :



On remarque qu'après une itération, le meilleur génome est présent dans la population. De fait, lors du processus de sélection suivant, il y a de fortes chances que ce génome soit sélectionné, éventuellement plusieurs fois. Par conséquent, la population va avoir tendance à s'uniformiser pour contenir de plus en plus de génomes « idéaux » (une certaine diversité de la population sera assurée par les effets des mutations). Au bout de plusieurs générations, on est donc assuré que le meilleur individu possède un génome très adapté à son environnement. De plus, le fait d'explorer plusieurs zones – par le biais de la mise en jeu de plusieurs génomes initialement choisis aléatoirement – de l'espace des solutions permet de trouver plus facilement un maximum global.

3.3 L'application au cas d'une fonction d'évaluation

Voici quelques pistes pour l'application de l'idée des algorithmes génétiques pour l'apprentissage automatique d'une fonction d'évaluation pour le jeu d'Othello.

En supposant pour l'instant qu'une seule fonction d'évaluation soit utilisée tout au long de la partie, on sait que la qualité d'une position est calculée de la manière suivante :

$$f(p) = \alpha_1 c_1(p) + \ldots + \alpha_m c_m(p)$$

et que la question qui se pose concerne la détermination automatique des coefficients $\alpha_1, \ldots, \alpha_m$. Ces coefficients doivent permettre au programme qui les utilise d'avoir un niveau de jeu élevé. En d'autres termes, le problème de la détermination des coefficients est celui de l'apprentissage de m réels permettant de maximiser la qualité du joueur artificiel obtenu.

En termes d'algorithmes génétiques, on peut donc recourir à la modélisation naturelle suivante :

génome : un génome est un vecteur de m réels, chacun d'eux correspondant au poids (i.e. à l'importance) des critères mis en jeu ; une bonne stratégie est de définir à priori des valeurs maximales et minimales pour chacun des coefficients ;

population initiale : la population initiale est donc constitué de n génomes choisis au hasard, chaque génome ayant ses coefficients choisis entre les bornes définies *a priori* ;

fitness et environnement : l'environnement sera constitué par un ensemble de k génomes initialement choisis aléatoirement ; ces génomes permettent de mesurer la fitness de tous les génomes de la population : il suffit pour chaque génome de faire jouer le programme qui lui correspond contre ces k génomes constituant l'environnement et de compter le ratio de victoires (par rapport au nombre total de parties jouées) de ce génome ; c'est une mesure de fitness assez naturelle puisqu'elle mesure la qualité d'un joueur et que ce que l'on désire est l'augmentation de cette qualité (et donc de la fitness) ;

reproduction : le principe de reproduction suivant peut être implémenté : étant donné deux génomes $\mathbf{g}$ et $\mathbf{g}'$ et les coefficients respectifs $\alpha_1, \ldots, \alpha_m$ et $\alpha'_1, \ldots, \alpha'_m$, s'il est décidé que le croisement entre $\mathbf{g}$ et $\mathbf{g}'$ a lieu alors pour chaque critère, on tire deux nombres réels aléatoires entre les valeurs de ces critères pour les parents, chacun de ces nombres correspondant au gène étudié pour chaque enfant ; dans le cas d'une mutation, on peut décider d'altérer un gène en tirant un nombre aléatoire entre les bornes possibles du critère étudié. Ce processus de reproduction s'illustre en partie de la manière suivante : si $\mathbf{h}$ et $\mathbf{h}'$ sont les deux enfants de $\mathbf{g}$ et $\mathbf{g}'$ alors le premier gène de $\mathbf{h}$ est un nombre aléatoire entre α_1 et α'_1 et de même pour $\mathbf{h}'$; si le premier gène de $\mathbf{h}$ doit subir une mutation, il suffit de remplacer le premier coefficient correspondant par une valeur aléatoire comprise entre $\alpha_1^{\max}$ et $\alpha_1^{\min}$, qui sont les bornes maximales et minimales du premier gène.

renouvellement de l'environnement : il peut être décidé de renouveler l'environnement une fois toutes les T générations en considérant les k meilleurs génomes de la dernière génération. Dès lors, il est nécessaire de générer à nouveau une population aléatoire de génomes que l'on fera évoluer dans l'environnement ainsi obtenu.

4 Autres combinaisons

Dans ce document, la forme de fonction d'évaluation proposée est celle d'une combinaison linéaire de critères. Ce type de combinaison à l'avantage d'être très rapide à calculer, une fois les critères estimés. Il peut cependant s'avérer utile d'utiliser des combinaisons plus évoluées qu'une simple combinaison linéaire. Une première manière de faire cela, qui ne s'éloigne néanmoins que très peu d'un combinaison linéaire simple est de considérer une fonction dite « logistique » et de calculer la sortie de la fonction d'évaluation pour une position p comme :

$$q(p) = \frac{1}{1 + \exp(-\alpha_1 c_1(p) - \ldots - \alpha_m c_m(p))}.$$

On s'aperçoit que l'argument de la fonction exponentielle apparaissant au dénominateur de cette fonction est précisément une combinaison linéaire de critères. Si la combinaison $\alpha_1 c_1(p) + \ldots + \alpha_m c_m(p)$ linéaire a une valeur élevée alors la fonction q précédente a une valeur proche de 1 ; si en revanche la combinaison linéaire prend une valeur négative, la fonction q prend une valeur proche de 0. (Sous certaines conditions, la valeur $q(p)$ peut être interprétée comme la probabilité que la position p soit une position gagnante.) L'utilisation d'une telle fonction peut éventuellement faciliter l'apprentissage, en permettant notamment de fournir des bornes assez lâches sur les valeurs extrêmes des différents gènes.

Il est par ailleurs possible d'envisager des modes de combinaison différents pour les critères. Une combinaison polynomiale, faisant intervenir la valeur des critères mesurés par le biais de certaines de leurs puissances est une première approche se distinguant complètement d'une combinaison linéaire. Le problème de la détermination des bonnes puissances à sélectionner, en plus des paramètres multiplicatifs qui y sont associés, est toutefois beaucoup plus difficile que dans le cas linéaire (le nombre de paramètres à otpimiser est beaucoup plus grand).

Une autre stratégie, certainement plus performante est celle qui consiste à considérer les valeurs des critères mesurés sur une position donnée comme les entrées d'un vecteur à traiter par un

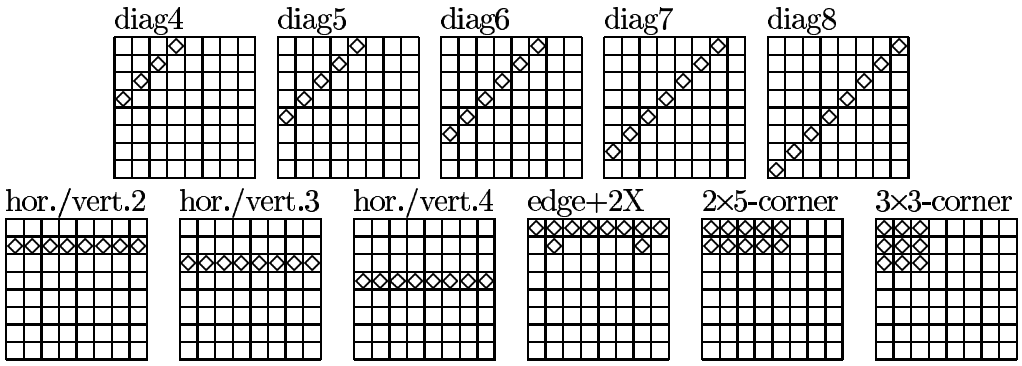


Figure 2: Motifs utilisés par Logistello (images récupérées de [Buro, 2000]).

réseau de neurones multi-couches. Ce type d'architecture, qui s'inspire initialement des réseaux de neurones présents dans le cerveau, est l'une famille de modèles qui a été beaucoup étudiée dans à la fin des années 80 et au début des années 90. Les perceptrons multi-couches se sont montrés particulièrement efficaces pour des tâches de reconnaissance de motifs complexes et ont été utilisés avec succès pour le jeu de BackGammon, la reconnaissance de chiffres manuscrits. . . Le principal désavantage de ce type de modèle est que le temps de traitement d'un vecteur est particulièrement long par rapport à une simple combinaison linéaire. Il est par ailleurs particulièrment compliqué de déterminer la meilleure architecture, i.e. le nombre de neurones à utiliser. Il n'est pas possible de donner ici une description exhaustive de ce type de modèles. Le web regorge néanmoins d'une multitude d'information sur le sujet et quelques documents peuvent être par ailleurs transmis aux étudiants intéressés.

5 Pour aller plus loin : utilisation de motifs

Une autre stratégie pour l'élaboration d'une fonction d'évaluation est celle proposée par M. Buro [Buro, 2000] (voir également ses nombreuses contributions sur sa page). Elle repose sur l'observation qu'une position de jeu en Othello peut être visualisée comme un ensemble de configurations locales qui s'appuient sur des « motifs » de différentes formes et situés à différents endroits du plateau de jeu (cf. Figure 2). A chaque configuration de pions au sein d'un motif est associée une note calculée en fonction de la fréquence d'apparition de cette configuration (dans le motif précisément étudié) dans des parties gagnantes parmi de nombreuses (plusieurs dizaines de milliers pour Logistello) parties d'Othello jouées par des experts. En utilisant une combinaison linéaire des notes attribuées à chaque configuration de motif, une note globale de la position est alors calculée. (De façon plus précise, les coefficients de la combinaison linéaire sont « appris » à l'aide de la méthode de régression logisitique.)

Pour faire une analogie avec le système de fonction d'évaluation à base de critères décrit précédemment les critères correspondent dans cette approche à base de motifs à l'apparition ou non d'une configuration donnée au sein d'un motif local ; l'importance d'un critère est mesurée par la note associée à la configuration (i.e. la fréquence de victoires obtenue lorsque la configuration locale apparaît).

Deux caractéristiques particulièrement notables de cette approche sont les suivantes : d'une part, la méthode d'attribution d'une note à une configuration est très simple et relie directement l'occurence d'une configuration avec l'issue finale de la partie, et, d'autre part, une programmation judicieuse permet de disposer d'une fonction d'évaluation où tout repose sur l'accès à des éléments de tableau, ce qui rend cette fonction d'évaluation d'une grande efficacité (et permet très facilement, sans modification de l'algorithme AlphaBeta de faire des recherches à une profondeur de 12 à 16 sur un ordinateur doté d'une puissance de calcul usuelle). Cette techniques des motifs demande néanmoins une programmation précautionneuse et précise de la gestion des tableaux (il suffit qu'une petite erreur d'indice pour que la fonction d'évaluation n'ait plus aucun sens).

6 Conclusion

La conception d'une fonction d'évaluation de qualité et qui présente en même temps des caractéristiques de rapidité de calcul constitue le point délicat de la réalisation d'un jeu de réflexion reposant sur MinMax/AlphaBeta. Cet aspect est celui qui le fait le plus appel à l'inventivité de chacun ainsi qu'à sa technique de programmation. Il est néanmoins assez facile de créer une fonction d'évaluation performante, en réglant par exemple les paramètres à la main. La mise en œuvre d'une méthode d'optimisation par algorithme génétique ou bien encore par apprentissage par renforcement (non couvert dans ce document mais qui a notamment fait ses preuves pour le jeu de Backgammon) permet cependant d'obtenir très facilement et automatiquement des programmes de grande qualité. Si l'algorithme génétique est bien programmé, la qualité des résultats peut même être très surprenante.

Références

[Buro, 2000] Buro, M. (2000). Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello. In *Games in AI Research*.