Informatique

L'apprentissage des ouvertures chez Logistello

Par Michael Buro

La force des programmes jouant à des jeux de réflexion a considérablement augmenté ces dernières années. Cela a été rendu possible, d'une part, par l'augmentation de la profondeur de recherche due à l'accéleration du matériel et aux raffinements des algorithmes de recherche dans les arbres et, d'autre part, par la mise au point de meilleures fonctions d'évaluation qui sont capables d'estimer les chances de gains à la fois précisement et rapidement. Dans les jeux tels que les Échecs, le Go ou Othello, la phase d'ouverture est très importante. Or c'est là que repose une des faiblesses connues des algoritmes de jeu, sans doute due à la difficulté de concevoir des plans stratégiques. D'où l'utilisation des bibliothèques d'ouvertures pour circonvenir à ce problème. Jusqu'à récemment on n'attachait pas beaucoup d'intérêt à la génération automatique de telles bibliothèques, dans la mesure où on pouvait trouver les « bonnes » suites de coups dans la littérature, les rentrer manuellement dans sa machine en vérifiant éventuellement que son programme était capable de dominer les complications tactiques qui en résultaient, et en changer si nécessaire après chaque partie perdue. Cependant, la mise en œuvre de serveurs de jeux (en particulier sur Internet) sur lesquels les programmes connectés peuvent jouer contre d'autres programmes ou des humains toute la journée, et la multiplication du nombre des parties qui en a résulté, ont rendu nécessaire que les programmes puissent modifier dynamiquement leur répertoire d'ouvertures.

Cet article décrit un algorithme qui trouve des alternatives raisonnables dans l'ouverture. Il est basé sur une idée simple mais efficace : refuser de perdre deux fois la même partie. De plus, il est même capable, si on le laisse tourner assez longtemps, de battre un adversaire plus fort qui n'aurait pas ses capacités d'apprentissage, voire de corriger automatiquement une bibliothèque d'ouvertures existante en en trouvant les faiblesses.

Stratégies pour une suite de parties.

Si un joueur électronique veut non seulement pouvoir être efficace dans une partie précise contre un adversaire dont il ne sait rien, mais aussi dans une séquence de parties contre cet adversaire, il devra sans doute résoudre les problèmes particuliers que posent les stratégies de match, simples mais néanmoins efficaces, que ne peuvent pas contrer les techniques connues de recherche dans les arbres de jeu (minimax, alpha-bêta, etc.). La plus évidente et la plus simple de ces stratégies est la suivante :

I. Essayer de rejouer les parties gagnées.

Un programme dépourvu de mécanisme d'apprentissage et déterministe suit cette stratégie, mais, d'un autre côté, en est en même temps victime, puisqu'il ne sait pas dévier des parties qu'il a perdues. La première idée pour remédier à ce problème est l'utilisation d'une bibliothèque d'ouverture avec une composante aléatoire pour choisir les coups. Mais cette approche n'est pas très flexible, ni d'un grand intérêt sur le plan théorique. De plus, ce n'est pas vraiment une solution puisque la probabilité de perdre augmente après chaque défaite (si l'adversaire suit la stratégie I). Pour contrer la stratégie I, il est nécessaire de savoir trouver de bonnes alternatives. Une manière de le faire, passive mais qui marche néanmoins, est de suivre le principe suivant :

II. Essayer de replacer à l'adversaire les variantes avec lesquelles il vous a battu.

L'idée derrière cette méthode élégante est, en quelque sorte, de laisser l'adversaire vous montrer vos propres fautes afin de jouer vous-même la prochaine fois les coups gagnants qu'il aura trouvés. Même un adversaire largement plus fort, mais dépourvu de mécanisme d'apprentissage, peut être mis en difficulté de cette manière puisque, à terme, il jouera contre lui-même. Comment peut-on battre la stratégie II ? Pour se montrer supérieur, il est nécessaire d'être inventif, pour surprendre l'adversaire dans les parties suivantes. Ce qui mène finalement au troisième principe :

III. Si une position a déjà été rencontrée mais qu'aucun coup gagnant n'est connu, alors chercher des coups de rechange prometteurs.

Un programme qui suit ces stratégies est un adversaire très déplaisant à jouer, dans la mesure où il essaye de replacer ses victoires et celles que vous lui avez infligées, où il ne perd jamais deux fois de la même manière, où il apprend les grands schémas d'ouverture qu'il n'aurait peut-être jamais trouvés par lui-même, où, enfin, il corrige ses erreurs dans les parties successives. De plus, un tel programme peut vérifier la qualité de ses innovations avant les tournois en jouant contre lui-même.

L'algorithme de génération de la bibliothèque d'ouvertures

Les stratégies I et II peuvent être implémentées facilement : si le programme garde trace de toutes ses parties perdues jusqu'à un instant donné, il peut, à partir d'elles, reconstruire un arbre de jeu dans lequel les résultats des parties sont propagées des feuilles vers la racine suivant le principe du Minimax. Quand il joue une partie, il lui suffit alors de rechercher dans l'arbre ainsi construit la position courante : si elle a déjà été rencontrée et qu'elle est étiquetée comme gagnante, on connait un coup gagnant dans l'arbre. Sinon, c'est soit qu'elle est nouvelle (auquel cas le programme effectue une recherche alpha-bêta normale pour trouver un coup), soit qu'elle est connue mais étiquetée comme perdante ou nulle. Dans ce cas, la stratégie III indique qu'il faut regarder s'il n'y a pas des alternatives plus prometteuses. Bien sûr, ce procédé ne doit pas concerner que la position en cours puisque la faute peut avoir été commise plus tard. Il est par conséquent nécessaire de connaître les évaluations heuristiques des coups qui ne sont pas encore joués pour effectuer le meilleur choix global.

Pour faire marcher concrètement l'algorithme, on définit (récursivement) ci-dessous la *valeur déviante* V_D d'une position : c'est la valeur de la variante que suivrait, à partir de cette position, deux joueurs appliquant les stratégies I, II et III.

Étant donné un arbre de jeu T construit à partir de toutes les parties déja jouées, notons $V_T(v) \in \{Perte, Nulle, Gain\}$ *la valeur minimax de la position v (pour la couleur qui a le trait) découlant des résultats des parties de T, S(v) l'ensemble des successeurs de v dans T et S'(v) l'ensemble des successeurs de v qui ne sont pas dans T.*
Alors la valeur déviante $V_D(v)$ *d'une position est définie comme suit :*

- *si v n'est pas dans T alors* $V_D(v)$ *est l'évaluation heuristique de v.*

- *pour une position terminale v dans T,*

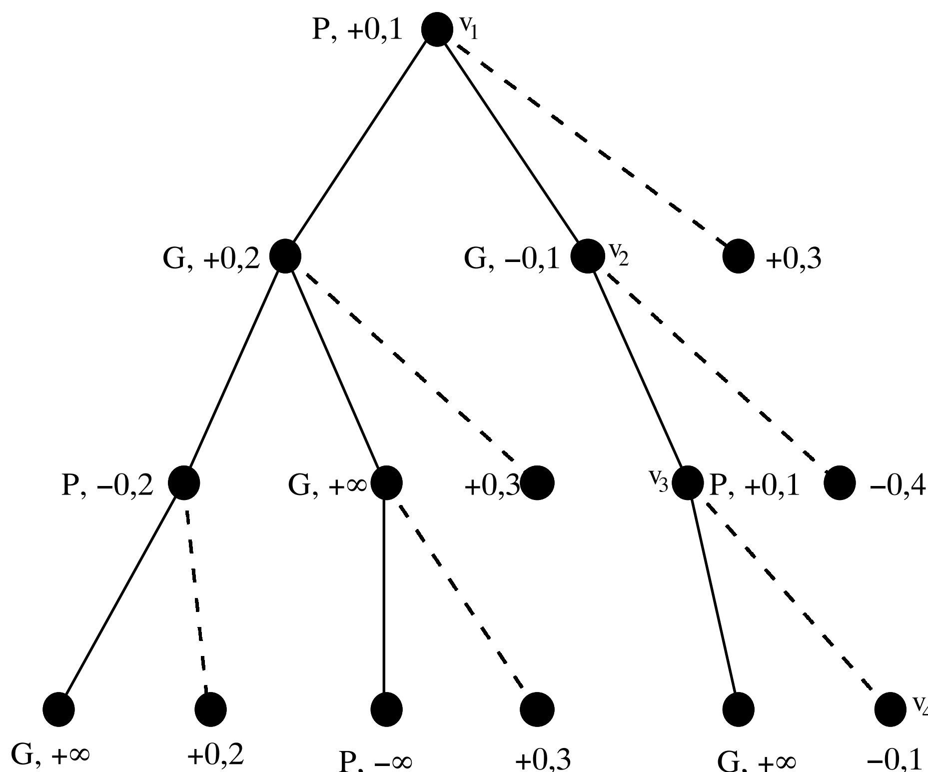
  si $V_T(v) = Gain$, $V_D(v) = +\infty$
  si $V_T(v) = Nulle$, $V_D(v) = 0$
  si $V_T(v) = Perte$, $V_D(v) = -\infty$

  on modélise ainsi le résultat exact de la partie.

- *si v n'est pas une position terminale dans T, alors*

  si $V_T(v) = Gain$, $V_D(v) = \max\{-V_D(w) \mid w \in S(v) \text{ et } V_T(w) = Perte\}$
  si $V_T(v) = Nulle$, $V_D(v) = \max\{-V_D(w) \mid w \in S(v) \text{ et } V_T(w) = Nulle \text{ ou } w \in S'(v)\}$
  si $V_T(v) = Perte$, $V_D(v) = \max\{-V_D(w) \mid w \in S(v) \text{ et } V_T(w) = Gain \text{ ou } w \in S'(v)\}$

  Dans chacun des trois cas, la valeur déviante est calculée récursivement à partir des successeurs de v qui ont mené aux meilleurs résultats dans les parties réelles. Cela assure de suivre la stratégie II. Si la position considérée n'a mené qu'à des nulles ou à des pertes, alors on regarde en plus dans toutes les alternatives pour choisir heuristiquement la meilleure suite.



La figure montre un exemple d'arbre de jeu T construit à partir de trois parties. Les positions sont étiquetées, d'une part, par les valeurs minimax $V_T(v)$ (G pour Gain, N pour Nulle et P pour Perte) issue de T et, d'autre part, par les valeurs déviantes $V_D(v)$, les alternatives heuristiques étant dessinées avec des pointillés. À la position racine v_1 l'algorithme doit choisir entre les deux coups « perdants » et l'alternative heuristique. Si l'on suppose que les deux joueurs suivent les stratégies décrites plus haut, alors la séquence optimale est $v_1,\ldots, v_4$. À cette position v_4 le programme considère qu'il y a de l'espoir puisque l'évaluation heuristique donne –0,1 pour l'adversaire. Par conséquent, en v_1 il va choisir le second coup « perdant » et dévier des lignes connues en v_3.

Avant d'utiliser cet algorithme, il est nécessaire de répondre à deux questions : quelles parties utiliser pour construire l'arbre T, et comment évaluer les alternatives ?

La réponse à la seconde question est claire. Il faut bien évidemment avoir une évaluation qui permettent de comparer les notes de différentes phases de la partie : le mieux semble être d'avoir une fonction d'évaluation dont l'interprétation soit globale, par exemple une estimation de la probabilité de gain ou de l'espérance de la différence de pions finale. Ceci posé, on peut, pour chaque position de la partie en cours d'analyse, chercher les notes (heuristiques) des alternatives éventuelles en faisant une recherche normale de milieu de partie, de préference à une profondeur au moins égale à celle que l'on atteint en tournoi pour éviter les mauvaises surprises.

La première question pose des problèmes plus subtils. Pour les parties perdues ou nulles, pas de problème : elles seront analysées et incluses dans l'arbre sans aucune restriction, puisque les premières seront répétées avec couleurs inversées, tandis que les secondes ne devront pas être rejouées contre un adversaire plus faible. Mais que faire des parties gagnées ? Afin d'éviter l'évaluation de nombreuses positions claires, il faut tester, avant d'importer une partie gagnée, si l'adversaire n'a pas eu, à un moment de la partie, une possibilité prometteuse (*i.e.* avec une forte note), auquel cas la partie doit être examinée puisque c'est une défaite potentielle. De telles parties peuvent être incluses dans l'arbre d'ouverture et corrigées en faisant jouer le programme contre lui-même. Cette pratique tend cependant à biaiser la bibliothèque d'ouvertures, dans la mesure où seules les parties les plus récentes sont rejouées. On peut, pour réduire cet effet néfaste, rechercher de temps en temps dans toute la bibliothèque d'ouvertures les positions « suspectes » qui ne sont pas gagnantes mais auxquelles existent de bonnes alternatives : les parties correspondantes seront finies pour les corriger.

Discussion

Nous avons présenté dans cet article un algorithme qui permet d'apprendre à partir de ses parties, de telle sorte que des variantes raisonnables peuvent être trouvées pour éviter de perdre deux fois la même partie, et que même des adversaires plus forts peuvent être mis en danger en utilisant leurs coups gagnants. Le temps nécessaire pour évaluer les alternatives possibles dans une partie est du même ordre de grandeur que celui pris pour la partie elle-même si la même profondeur est utilisé, et est, par conséquent, acceptable.

En pratique, cette nouvelle technique est implementée dans Logistello, l'un des plus forts programmes d'Othello actuels, sous une forme cependant légérement modifiée : Logistello joue pour gagner, et donc compte les nulles comme des défaites pour éviter leur répétition. Connecté au serveur Internet d'Othello (IOS), il a démontré ces derniers mois sa force au cours de centaines de parties contre d'autres programmes qui étaient aussi équipés de procédés d'apprentissage, en leur resservant leurs propres coups ou innovations.

Remerciements

Je tiens à remercier toutes les personnes qui ont pris part, sur IOS, aux discussions sur les mécanismes d'apprentissage. Les défaites de Logistello contre les forts programmes de l'IOS ont motivé ce travail.

Traduction : Stéphane Nicolet

Références

M. Buro, « *Techniken für die Bewertung von Spielsituationen anhand von Beispielen* », thèse de Ph.D., 1994, Université de Paderborn, Allemagne.

A. L. Samuel, « Some studies in marchine learning using the game of checkers », *IBM Journal of Research and Development* **3** (1959), 210-229.

T. Scherzer, L. Scherzer, D. Tjaden, « Learning in Bebe », *in* T. A. Marsland et J. Schaeffer (eds.), *Computer, Chess and Cognition*, Springer-Verlag (1990).

Partie Internet 1993

	a	b	c	d	e	f	g	h
1	49	18	14	33	13	19	46	51
2	52	24	15	12	11	16	47	50
3	32	22	3	4	7	8	26	36
4	31	20	5			6	9	38
5	35	21	29			1	17	37
6	34	48	30	2	27	10	23	39
7	60	43	44	45	28	25	57	40
8	54	59	56	41	42	58	53	55

Logistello 38-26 Rev/Kitty