

Etat Actuel de la Programmation du Jeu de Go

Bruno Bouzy

LIP6, Université Paris 6, 4, place Jussieu,
75252 PARIS Cedex 05 FRANCE
Tél.: 01 44 27 70 10 Fax: 01 44 27 70 00
e-mail : bouzy@laforia.ibp.fr

Charles Mary

Le jeu de Go est un jeu très ancien aux règles simples

Le jeu de Go est un jeu d'origine chinoise vieux de trois à quatre mille ans qui est très populaire au Japon, en Chine et en Corée où il existe des joueurs professionnels. Il ne s'est développé que récemment aux États-Unis et en Europe où il n'existe que de forts amateurs. Ses règles sont très simples mais sa complexité est immense. Le lecteur novice souhaitant connaître les règles de ce jeu peut se reporter à l'annexe A.

La combinatoire du Go est plus grande que celle du jeu d'Échecs

Il a d'abord été difficile d'évaluer théoriquement la complexité algorithmique du jeu de Go. En 1983, Robson a prouvé que le problème consistant à décider si un joueur peut forcer la victoire à partir d'une configuration donnée était un problème dont le temps de résolution est une fonction exponentielle de la taille du goban. Il existe d'autres résultats théoriques sur la complexité algorithmique de ce jeu qui tendent tous à dire que le problème est encore plus complexe que ce que l'on pensait a priori! Pour se fixer les idées, Allis compare pratiquement la combinatoire du jeu de Go avec celle d'autres jeux, et en particulier avec celle du jeu d'Échecs. Le nombre de coups possibles dans une position typique avoisine en moyenne 200 au Go contre 35 aux Échecs, le nombre de coups joués dans une partie

typique avoisine 250 au Go contre 80 aux Échecs et enfin la taille très approximative de l'arbre des possibilités du jeu de Go est environ de 10^{600} alors que celui du jeu d'Échecs n'est seulement (!) que de 10^{70} environ.

Le meilleur programme de Go a seulement le niveau de débutant en club

Le précurseur de la programmation du jeu de Go fut l'américain Zobrist qui implémenta un modèle d'organisation visuelle basé sur des champs de potentiel en 1970. Dans les années 70, Wilcox et Reitman développèrent le programme Interim.2 basé sur la reconnaissance des formes. Son niveau devait être environ 25 ou 30ème kyu... Depuis cette période, des articles de recherche sur le sujet paraissent dans les revues internationales d'informatique et d'intelligence artificielle. En 1987, une revue, Computer Go, s'est spécialisée dans ce domaine. Depuis quelques années, plusieurs tournois d'ordinateurs sont organisés. Le tournoi le plus prestigieux est organisé par l'industriel taïwanais M. Ing au mois de novembre de chaque année. Dans les années 80, celui-ci a promis un million de dollars au programme qui battra un joueur professionnel dans une partie à égalité avant l'an 2000. Désormais, le risque est faible car en 1991, Goliath, le programme hollandais de Mark Boon perdait à 15 pierres de handicap contre de forts joueurs amateurs. En novembre 1996, Handtalk, le programme chinois de M. Chen qui s'est imposé dans le tournoi des ordinateurs, a perdu une partie à 11 pierres de handicap après avoir gagné à

15 et 13 pierres les années précédentes. Le niveau de Handtalk n'est plus celui d'un joueur de club complètement débutant. Nous l'évaluons entre 8 et 10ème kyu. D'autres compétitions d'ordinateurs existent depuis peu. La FOST Cup est un championnat ouvert à tous les programmes et organisé chaque année par les Japonais. Sur Internet, les programmes peuvent jouer sur l'*Internet Go Server* (IGS) et être classés sur la *Computer Go Ladder*. Ainsi, les confrontations et échanges d'idées entre les programmeurs sont largement facilités.

La machine résout des sous-problèmes posés par le jeu de Go comme les joueurs très forts, voire mieux

Au début des années 60, Thorp et Walden ont effectué des simulations sur ordinateur pour analyser le jeu. Ils ont trouvé des stratégies applicables sur des petits gobans. En 1976, Benson a proposé un formalisme dans lequel il a démontré des théorèmes très simples sur les problèmes de vie et mort (tsumegos). Les théorèmes de vie et mort de Benson sont utilisés par la plupart des programmes actuels. Depuis 1990, Thomas Wolf a développé un programme, GoTools, de résolution et de génération de problèmes de tsumego qui atteint des performances du niveau de forts joueurs amateurs. Ce résultat est remarquable comparé aux résultats des programmes jouant une partie complète. Mieux, GoTools a trouvés des erreurs dans certains dictionnaires de problèmes. Un résultat très intéressant est paru en 1991 en mathématiques appliquées. Elwin Berlekamp a appliqué la théorie des jeux sommables et indépendants de John Conway au petit yose (la toute fin de partie de Go). Lorsque la partie devient décomposable en jeux indépendants, la théorie proposée permet de trouver la séquence optimale jusqu'à la fin de la partie. Certaines de ces séquences ne sont pas toujours trouvées par des joueurs humains professionnels! A chaque fois, les résultats obtenus, même s'ils ouvrent des horizons nouveaux, ne s'appliquent qu'à des sous-problèmes bien définis et limités du jeu de Go (un problème de

tsumego, le petit yose) mais en aucun cas à sa globalité.

Le jeu de Go est un terrain d'études privilégié pour les sciences cognitives

Dans le domaine des sciences cognitives, des expériences sur des joueurs de Go ont été faites en 1976 par Judith Reitman. En observant temporellement la façon de reproduire des gobans vus préalablement par des joueurs débutants ou experts, on voulait essayer de déduire comment le joueur de Go stockait et réutilisait ses connaissances visuelles. Les conclusions furent similaires à celles faites aux Échecs par Chase et Simon: les experts et les débutants utilisent une capacité mémoire comparable mais les premiers mémorisent une information plus spécialisée que les seconds. En 1995, nous avons élaboré un modèle cognitif du joueur de Go que nous avons validé en implémentant le modèle dans le programme de Go Indigo. Cette expérience nous a essentiellement appris que les joueurs de Go utilisent une forte majorité de connaissances non conscientes. Il est alors difficile d'explicitier ces connaissances pour les donner à une machine. C'est d'ailleurs une des raisons qui limitent le niveau des programmes de Go actuels.

La programmation d'une partie de Go complète est très difficile

La programmation du jeu de Go oblige les programmeurs à sortir des sentiers battus et à innover. Le découpage archaïque d'un programme en recherche arborescente, génération de coups et fonction d'évaluation, hérité de la programmation des Échecs, doit être remis en cause. Malgré les efforts consentis depuis 1970, la programmation du jeu de Go se trouve encore dans un état préhistorique. Chaque programmeur essaie les techniques qu'il juge prometteuses : la reconnaissance de formes, des modèles d'influence, la recherche arborescente sur des sous-butts locaux, des fonctions d'évaluation globales plus ou moins complexes, la programmation orientée objet, des techniques d'acquisition de connaissances et d'apprentissage. Chaque programmeur

trouve le juste équilibre entre ces techniques pour mettre au point son système. Jusqu'à maintenant, aucune solution à elle seule n'a fait de miracle ou l'unanimité, loin de là. Le problème se situe dans l'intégration de ces différentes techniques. Il est possible de dégager les principales lignes directrices d'un programme de Go. D'abord, la complexité arborescente étant ce qu'elle est, l'évaluation d'une position, passe :

- par une décomposition en plusieurs sous-positions locales,
- par des calculs sur ces sous-positions,
- et enfin par une recombinaison.

La décomposition est basée sur l'instanciation d'objets suivant les concepts du domaine. Ceux-ci sont nombreux : les "groupes", les "zones", les "territoires", les "connexions", les "contacts", les "séparateurs", les "yeux" en sont des exemples simples. Cette diversité fait que, voulant éviter la complexité combinatoire du jeu, on aboutit, de fait, à une complexité "ontologique" : une position contient un nombre et une richesse d'objets absolument ahurissants. Les descriptions de la position étant malheureusement trop nombreuses, la décomposition doit alors être faite suivant un point de vue particulier. Le programme perd donc de l'information avec cette décomposition initiale. Pour chaque sous-position locale ou objectif particulier (capturer un groupe, enlever des yeux, agrandir un territoire), un calcul est effectué. Si le calcul se termine, son résultat est enregistré simplement. Quand le calcul est trop long, il est interrompu et la sous-position n'est évaluée qu'avec incertitude. La rationalité limitée des programmes et la limite imposée par la puissance des machines engendre une incertitude de fait, qui doit être gérée par la recombinaison. Celle-ci est également approximative. Premièrement, comment comparer un gain de territoire avec une capture de groupe ? L'hétérogénéité des objets manipulés rend la synthèse des résultats locaux problématique. Par contre, l'être humain, avec ses facultés de "jugement", n'est pas gêné par cette difficulté. De plus, la recombinaison suppose que les calculs

effectués sont indépendants. Ce qui n'est pas le cas en réalité : un coup paraissant inutile à deux sous-positions prises séparément peut s'avérer utile si l'on considère les deux sous-positions dans leur ensemble. Ou bien, jouer un coup dans une sous-position peut améliorer cette sous-position mais détruire une sous-position voisine... On retrouve là le "frame problem" bien connu des chercheurs en intelligence artificielle. Ces choix d'architecture, nécessaires, engendrent des approximations qui s'ajoutent et font sensiblement baisser le niveau du programme.

Le temps de réponse et l'incrémentalité

Les programmes de Go suivent une spécification implicite qui consiste à jouer un coup en un temps raisonnable, c'est-à-dire en un temps comparable à celui des joueurs humains. En effet, psychologiquement, il est très ennuyeux de jouer contre un programme médiocre sur l'échelle humaine, et... lent. Cette contrainte de temps influe sur la conception du programme. Sur des gobans 19x19, il est souhaitable que le programme mette à jour ses données incrémentalement. Cela lui permet de ne pas recalculer des sous-positions situées suffisamment loin du dernier coup et ainsi d'économiser un temps précieux. Dans notre programme, nous avons mis au point un mécanisme d'"empreinte" qui modélise, pour chaque objet reconnu, l'ensemble des intersections sur lesquelles le coup joué modifie l'objet.

Les très bons résultats atteints sur les sous-problèmes ne sont pas suivis par des résultats identiques sur le jeu global

En ce qui concerne les problèmes de vie et de mort, GoTools, le programme de Thomas Wolf, suppose que les groupes étudiés sont complètement entourés par des groupes complètement vivants. En partie réelle, ces cas ne comptent que pour 1 ou 2%. Dans tous les autres cas, en plus de la faculté de faire deux yeux, la vie et la mort des groupes dépend de l'encerclement du groupe et... de la vie et la mort des groupes voisins. Le problème

peut être sans fin... Ce qui est intolérable. Il faut alors trouver des mécanismes pour stopper les calculs au bon moment, ce qui un problème ouvert.

Pour les toutes fins de partie, la théorie de Berlekamp permet de jouer des fins de partie très spécifiques en gagnant un point par rapport à ce que font les professionnels. Quand on sait que, en partie réelle, même les forts joueurs font parfois des erreurs de trente points, on se demande l'utilité d'une telle perfection dans un programme de Go qui fait couramment des erreurs de cent points! Plus sérieusement, il faut préciser que les positions spécifiques où s'appliquent la théorie de Berlekamp, supposent :

- que la décomposition, avec tous les problèmes qu'elle pose, soit faite sans erreur,
- que les calculs sur les sous-positions, également avec leur lot de problèmes, soient effectués jusqu'au bout,
- que les sous-positions soient indépendantes les une des autres, ce qui n'est pas le cas des positions combatives,
- que les coups ou séquences de coups trouvés ne remettent pas en cause ni la décomposition, ni les résultats de calculs locaux (vie et mort des groupes).

Ces quatre raisons montrent que la route est encore longue avant d'avoir un programme qui utilise cette théorie avec des résultats tangibles. Martin Müller, qui travaille à Berkeley avec Berlekamp, est le plus avancé dans l'intégration des modules utilisant cette théorie des jeux particulière dans un programme jouant une partie complète - Explorer en l'occurrence.

Une méthode pragmatique

Nous avons effectué une enquête auprès des principaux programmeurs de Go dans le monde pour connaître leur avis sur quelques points précis. Il s'avère que les programmeurs s'appuient à la fois sur de très bonnes idées et sur la mise au point et les tests. Une idée n'est bonne

que si elle donne de bons résultats en pratique. La méthode de programmation se rapproche plus de la méthode de prototypages successifs chère à l'intelligence artificielle et moins au cycle en V, avec spécifications externes, conception générale et détaillée, codage et tests, puis intégrations successives et recette. Les langages utilisés sont le C, le C++ (et même l'assembleur pour Handtalk!). On constate que la rapidité d'exécution est un souci pour les programmeurs au détriment des temps de développement. Ces programmes contiennent entre dix mille et cent mille lignes de code. Ils utilisent aussi éventuellement des langages spécifiques, pour la reconnaissance de patterns notamment, avec entre zéro et quinze mille règles. Le temps de développement est en moyenne de trois homme-années. Enfin, à la question de savoir dans combien de temps un programme battra un professionnel 9ème dan, les programmeurs ont des avis très divergents : si certains pensent que cela arrivera dès 2005, d'autres pensent que dans 200 ans l'homme sera toujours le plus fort...

En France, plusieurs travaux sont en cours

En 1992, à Montpellier, Pierre Pompidor a soutenu une thèse en apprentissage symbolique au jeu de Go à partir d'exemples et de contre-exemples et de concepts géométriques. Depuis 1991, Jacques Pitrat, directeur de recherche au CNRS, a encadré, et encadre encore, des thèses sur la programmation du jeu de Go. Nous avons écrit le programme de Go "Indigo" dans le cadre d'une modélisation cognitive du joueur de Go et nous prolongeons actuellement notre travail vers le raisonnement spatial et l'IA distribuée. Patrick Ricaud a modélisé la stratégie du début de partie en utilisant l'abstraction et écrit le programme "Gobelins". Tristan Cazenave a écrit le programme "Gogol", qui résout des problèmes, explique les raisons de ses coups, engendre et généralise des règles en logique des prédicats du premier ordre, compile ces règles... pour résoudre des problèmes... et ainsi de suite. Gogol et Indigo sont honorablement classés dans

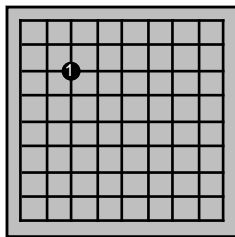
la Computer Go Ladder. Gogol a également participé à la FOST Cup en 1996.

Conclusion

Finalement, nous avons montré que le problème de la programmation du jeu de Go, très clair dans sa définition - les règles du jeu sont simples, le monde est clos - est en fait très complexe : le premier obstacle est combinatoire, le second est ontologique - ou cognitif, le troisième est pratique. Ce problème est passionnant : il est multi-points de vue, multi-techniques et il possède de nombreuses similitudes avec le monde réel. La programmation du jeu de Go passe par une analyse du fonctionnement cognitif du joueur de Go. On pourra alors représenter les connaissances humaines de façon à les utiliser efficacement grâce aux possibilités calculatoires des machines. Ce travail s'inscrit dans le cadre de la recherche en intelligence artificielle qui trouve dans la programmation du jeu de Go un terrain de recherche idéal et un défi difficile.

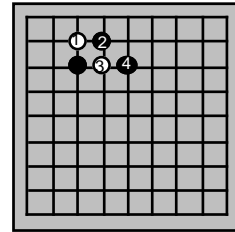
Annexe A: Aperçu des règles du jeu

Une partie se joue à deux joueurs, *Noir* et *Blanc* sur un damier appelé *goban* constitué de 19 lignes et 19 colonnes qui se coupent suivant 361 *intersections*. Pour simplifier, nous donnons un exemple de partie jouée sur 9-9 avec 81 intersections seulement. Au début de la partie le goban est *vide*. Noir commence en posant une *pièce* noire sur une intersection, par exemple avec ❶ :

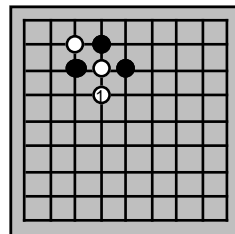


La pierre posée sur le goban est voisine de 4 intersections vides: les *libertés*. On ne peut pas jouer de coup sur une intersection où l'on n'a pas de liberté.

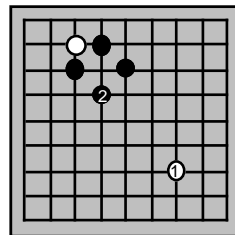
Une pierre posée sur une intersection du bord (respectivement coin) du goban possède 3 (resp. 2) libertés. Noir et Blanc posent à tour de rôle des pierres sur le goban.



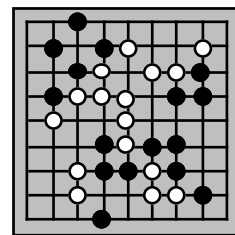
Ici, ❹ a fait *atari* à ❸ Cela signifie que ❸ ne possède plus qu'une seule liberté. A partir de cette position, deux cas sont possibles. Si Blanc défend sa pierre avec ❶ :



Blanc constitue une *chaîne* de 2 pierres. La chaîne blanche a 3 libertés. Par contre, si Blanc joue ailleurs, Noir peut *capturer* la pierre blanche :

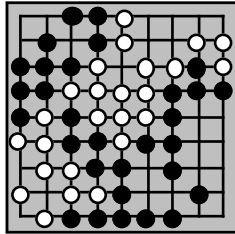


Les pierres capturées sont enlevées du goban. La partie se poursuit, le goban se remplit de pierres :



La partie se termine quand Noir et Blanc *passent*. A ce moment là, on compte le *score*. On attribue un *point* par

intersection *contrôlée* au joueur qui la contrôle. Une intersection est contrôlée par une couleur si elle est occupée par une pierre de la couleur ou si, étant vide, lorsqu'une pierre de l'autre couleur y sera posée, elle sera capturée. Le joueur qui a le plus de points *gagne* la partie.



Ici, toutes les intersections appartiennent soit à Noir, soit à Blanc. Noir contrôle 46 intersections et Blanc contrôle 35 intersections. Noir gagne de 11 points. Des joueurs de niveaux différents peuvent jouer ensemble de manière intéressante grâce au système du handicap. Au premier coup, le joueur le plus faible pose un nombre de pierres suffisant pour que l'issue de la partie soit incertaine. Les joueurs de Go sont échelonnés suivant une échelle de niveaux graduée en *kyus* et en *dans*. Un joueur totalement débutant est 30 ou 40ème kyu. Un joueur faible est 20ème kyu. Un joueur très moyen est 10ème kyu (bien que l'accès à ce niveau exige une passion et une persévérance). Un joueur fort est 1er kyu, puis 1er dan. Les meilleurs joueurs européens sont 6 ou 7ème dan et les meilleurs joueurs professionnels sont 9ème dan. Une différence de kyu ou de dans signifie que le joueur le plus fort donne autant de pierres d'avance au joueur le plus faible. Contrairement à d'autres jeux, ce mécanisme de classement et de handicap permet de motiver les débutants qui voient explicitement leur progrès et qui peuvent jouer contre des joueurs plus forts avec un espoir de gain.

Bibliographie

Quelques références bibliographiques pour le lecteur désireux de connaître la programmation du jeu de Go, plus en détails.

- Allis L.V., *Searching for Solutions in Games and Artificial Intelligence*, Ph.Thesis, Vrije Universitat Amsterdam, Maastricht, September 1994
<http://www.cs.vu/~victor/thesis.html>
- B. Bouzy : *Modélisation cognitive du joueur de Go*, Thèse de l'université Paris 6, 13 janvier 1995.
<http://www-laforia.ibp.fr/~bouzy>
- T. Cazenave: *Système d'apprentissage par auto-observation. Application au jeu de Go*, Thèse de l'université Paris 6, 13 décembre 1996.
<http://www-laforia.ibp.fr/~cazenave/Tristan.html>
- *The Computer Go page of the American Go Association.*
<http://www.usgo.org/computer/>
- Conway J., Berlekamp E.R., Guy R., *Winnings ways*, Academic press, 1982.
- M. Müller: *Computer Go as a Sum of Local Games*, Ph.D.Thesis, ETH Zürich, février 1995.
<http://nobi.ethz.ch/~martin/martin.html>
- Petersen E., *The Computer Go Ladder*,
<http://cgl.ucsf.edu/go/ladder.html>
- P. Ricaud: *Une approche pragmatique de l'abstraction appliquée à la modélisation de la stratégie élémentaire du jeu de Go*, Thèse de l'université Paris 6, 16 décembre 1995.
- Wolf T., The program GoTools and its computer-generated tsume go database, *Proceedings of the First Game Programming Workshop in Japan*, pp. 84-96, Hakone, 1994.
<http://www.qmw.ac.uk/~ugah600/gotools>