Vers un modèle computationnel du joueur de go

Bruno Bouzy

Université Paris 5, UFR de mathématiques et d'informatique, C.R.I.P.5,
email: bouzy@math-info.univ-paris5.fr,
http: www.math-info.univ-paris5.fr/∼bouzy/

Habilitation à diriger des recherches

soutenue le 26 novembre 2004

Rapporteurs du mémoire:

Keh-Hsun Chen, Professeur d'informatique, University of North Carolina,
Patrick Greussay, Professeur d'informatique, Université Paris 8,
Bernard Victorri, Directeur de recherche en informatique, CNRS.

Jury de soutenance:

Patrick Greussay, Professeur d'informatique, Université Paris 8,
Jaap van den Herik, Professeur d'informatique, Universiteit Maastricht,
Gérard Ligozat, Professeur d'informatique, Université Paris 11,
Jean Michel, Directeur de recherche en mathématiques, CNRS,
Dominique Pastre, Professeur d'informatique, Université Paris 5,
Jacques Pitrat, Directeur de recherche émérite en informatique, CNRS,
Jean-Charles Pomerol, Professeur d'informatique, Université Paris 6,
Bernard Victorri, Directeur de recherche en informatique, CNRS.

Résumé

Ce document présente les recherches effectuées sur la programmation du jeu de go depuis 1997 à l'université Paris 5 en vue d'obtenir une habilitation à diriger des recherches. Les avancées dans la programmation du jeu de go et des jeux de réflexion en général témoignent des progrès de l'intelligence artificielle. Cependant, le jeu de go résiste à l'informatisation par la taille de l'arbre du jeu rendant impossible toute recherche arborescente globale et par la difficulté à trouver une bonne fonction d'évaluation. La recherche présentée dans le cadre du projet Indigo vise à obtenir un programme "aussi bon que possible" en publiant le résultat de ces recherches. L'approche suivie de 1997 à 2002 est basée sur les connaissances du domaine et regroupe plusieurs thèmes de l'IA. Pour chaque thème, IAD, sciences cognitives, logique floue, représentation de l'incertitude, jeux combinatoires, analyse rétrograde, recherche arborescente, fonction d'évaluation, raisonnement spatial, l'apport scientifique est présenté. En 2002, l'approche Monte Carlo a été démarrée et a donné des résultats prometteurs. Ce qui a entraîné naturellement l'implémentation d'une approche associant Monte Carlo et les connaissances du domaine en 2003. A son tour, cette association a donné des résultats encore meilleurs aux dernières olympiades d'ordinateurs à Graz en 2003. A l'avenir, l'approche Monte Carlo sera poursuivie et associée à des approches d'apprentissage bayésien ou par renforcement.

1 Introduction

Actuellement maître de conférences en informatique à l'université Paris 5, je travaille sur la programmation du jeu de go et celle des jeux de réflexion depuis treize ans. En effet, après avoir travaillé dans l'informatique industrielle de 1986 à 1990, et étant passionné par le jeu de go, j'ai choisi de faire une thèse sur la programmation du jeu de go, intitulée "Modélisation cognitive du joueur de go", obtenue en janvier 1995 [33] à l'université Paris 6 sous la direction de Jacques Pitrat. Durant cette thèse, une première version de mon programme qui se nomme Indigo a été développée. Depuis 1997, j'appartiens au groupe SBC (Systèmes à Bases de Connaissances) de l'équipe IAA (Intelligence Artificielle et Applications), dirigée par Dominique Pastre, au sein du CRIP5 (Centre de Recherche en Informatique de Paris 5), le laboratoire d'informatique dirigé par Jean-Marc Labat et associé à l'UFR de mathématiques et d'informatique dirigée par Dominique Seret. J'y ai continué activement mes recherches sur la programmation du jeu de go, ce qui a produit une seconde version d'Indigo beaucoup plus forte que la première. Tout au long de ce travail, le résultat de ces recherches a été décrit dans des publications nationales et internationales. Le but de ce document est de *décrire les recherches effectuées entre 1997 et le début de l'année 2004* afin d'obtenir une *habilitation à diriger des recherches.* L'habilitation à diriger des recherches est un diplôme national dont la finalité essentielle est de permettre l'accès au corps de professeur des universités [2]. Par la délivrance de ce diplôme, les universités reconnaissent un niveau scientifique élevé caractérisé par une démarche originale dans le domaine scientifique, la maîtrise d'une stratégie autonome de recherche scientifique, et la capacité à l'encadrement des jeunes chercheurs [2].

Pour décrire ces recherches et obtenir l'habilitation à diriger des recherches, plusieurs points de vues sont nécessaires. La partie 2 répond d'abord à des questions cruciales d'objectif et de méthode. Elle montre quelle a été la démarche originale suivie depuis 1997. En autre, elle explique pourquoi ce mémoire s'intitule "Vers un modèle computationnel du joueur de go". La partie 3 présente un historique des activités effectuées, période par période. Cette partie donne un aperçu synthétique et chronologique du déroulement global de ces recherches. Afin de montrer la maîtrise d'une stratégie autonome de de recherche scientifique, la partie 4 présente les publications dans les revues ou conférences en commentant brièvement les publications majeures et en classant l'ensemble des publications. Dans la partie 5, les activités sont présentées thème par thème, thème étant compris au sens de domaine de l'IA. Pour chaque thème, un état de l'art est éventuellement présenté, puis l'apport scientifique des publications relatives à ce thème est mis en avant. La partie 6 présente un état de l'art, très sélectif, de la programmation des jeux de réflexion. Tous les états de l'art présentés dans ce document permettent de faire le point sur des références qui m'ont intéressées depuis un dizaine d'années, et ils permettent de mettre en avant l'apport scientifique de ce travail. La partie 7 liste les activités diverses liées à la recherche, notamment les activités liées à l'encadrement d'étudiants ou de jeunes chercheurs (sous-parties 6.4 et 7.3). La partie 8 décrit les perspectives

de ce travail. Enfin, la conclusion résume l'ensemble de ces recherches afin de montrer comment les caractéristiques attendues d'une habilitation à diriger des recherches ont été atteintes.

La plupart des publications citées par ce document se trouvent accessibles par le Web, sous forme de "pré-prints", à l'adresse http ://www.math-info.univ-paris5.fr/∼bouzy/publications.html. Cela permet d'accéder à des points plus détaillés et des références mentionnés par ce document dont l'objectif est volontairement synthétique et global.

2 Contexte, méthode et objectif des recherches

Cette partie présente le contexte des recherches effectuées. Premièrement, elle présente l'intérêt de produire un programme de go pour l'IA. Puis, elle décrit la façon dont les progrès du programme sont mesurés. Ensuite, elle montre comment ces recherches sont valorisées au sein du monde académique. En expliquant la différence entre modèle cognitif et modèle computationnel, elle explique ensuite pourquoi le titre de ce mémoire est "Vers un modèle computationnel du joueur de go". Enfin, avant d'exposer la méthode de recherche utilisée, cette partie synthétise l'objectif des recherches.

2.1 Pourquoi un programme de go ?

Les jeux de réflexion sont des domaines de prédilection de l'informatique et de l'IA pour tester leurs méthodes ; ils témoignent de leurs avancées et de leurs difficultés [139]. La programmation des jeux est fortement liée au paradigme arborescent. Plus l'arbre du jeu est petit, mieux les techniques de recherches arborescentes réussissent et meilleurs sont les programmes [55]. Par exemple, à othello, jeu pas trop complexe en termes arborescents (10^{58}), le meilleur programme, Logistello [63], dépasse largement le meilleur humain. Aux échecs, plus complexe (10^{123}), les meilleurs programmes sont du même niveau que les meilleurs humains. En 1997, Kasparov a été battu par Deep Blue [65]. En 2003, Kasparov a fait match nul contre Fritz, un des meilleurs programmes d'échecs actuel. Au jeu de go (10^{700})[1], les meilleurs programmes ont le niveau de joueurs humains de force très moyenne [55]. Au go, la recherche arborescente globale ne marche pas, contrairement aux échecs ou à othello. Le go est donc un domaine résistant permettant à l'IA de faire progresser ses méthodes et d'illustrer certains de ces thèmes [57, 55]. Les thèmes ou méthodes de l'IA qui peuvent être étudiés au travers du développement d'un programme de go sont entre autres, la représentation des connaissances, l'acquisition automatique, l'apprentissage supervisé ou par renforcement, les réseaux de neurones, la recherche arborescente, l'analyse rétrograde, Monte Carlo, la morphologie mathématique, la représentation de l'incertitude, la logique floue ou encore le raisonnement spatial. La partie 5 décrit les thèmes qui ont déjà interagi avec les recherches effectuées et la partie 8 montre quels thèmes seront étudiés à l'avenir. L'objectif

[1] Un minorant de factorielle 361

de développer un programme de go, tel quel, ne donne pas nécéssairement une crédibilité aux méthodes et thèmes abordés. Pour donner une réelle crédibilité aux méthodes et thèmes abordés, il faut développer un programme de go *aussi bon que possible*. Ainsi, seules les méthodes donnant un bon programme de go seront gardées et publiées. Dans ce contexte, le but de notre projet est de faire un programme de go aussi bon que possible. Ce programme s'appelle Indigo. Il est accessible via sa page web [46]. Les deux articles permettant d'avoir un aperçu global du fonctionnement du programme lui-même sont [35, 51]. Les autres articles décrivent les thèmes et méthodes abordés par ces recherches, montrant l'intérêt du développement d'un programme de go pour l'informatique et l'IA.

2.2 Comment mesurer les progrès ?

Pour obtenir un programme *aussi bon que possible*, un fonctionnement annuel a été adopté. Chaque année se termine par la production d'une version stable de Indigo. Par exemple Indigo99 est la version de Indigo à la fin de 1999. Pour améliorer le programme de l'année précédente, une version de travail est créée et comparée régulièrement à la version précédente. Un comparaison consiste en une série de parties sur damiers de tailles différentes, 9x9, 13x13 et 19x19. Elle peut comprendre ou non des parties à handicaps différents. Le résultat d'une comparaison est soit un nombre de pierres de handicap gagnées, soit un nombre de points moyen de parties à égalité. L'écart-type des résultats de parties de go 19x19 étant environ 50, il faut environ une centaine de parties pour obtenir une précision de 5 points avec un niveau de confiance de 65% seulement. Une comparaison coûte donc un temps d'exécution machine relativement important. Le prix à payer est élevé mais il apporte la certitude, précieuse dans un domaine complexe, que le nouveau programme est meilleur que le précédent. Chaque année, les progrès réalisés "en interne" sont donc connus. Par exemple Indigo2000 est quatre pierres plus fort que Indigo99. Le vrai test se passe à l'extérieur lors des compétitions d'ordinateurs, lorsque Indigo joue contre d'autres programmes de conceptions différentes. Dans cette optique, l'objectif est de participer régulièrement à des compétitions internationales d'ordinateurs pour mesurer ces progrès et ceux de la concurrence. Par exemple, 2003 a été une très bonne année. En interne, Indigo2003 est 40 points meilleur que Indigo2002. Qui plus est, "en externe", Indigo 2003 a terminé 4ème sur 10 sur 9x9 et 5ème sur 11 sur 19x19 à Graz en novembre 2003 lors des olympiades d'ordinateurs [6], ce qui a confirmé les progrès face à la concurrence.

2.3 Valorisation de la recherche

Cette partie montre comment la valorisation du travail a évoluée peu à peu, depuis le domaine de la programmation du go vers le domaine universitaire. Pour cela, il est important de préciser les trois grandes catégories de programmeurs de go rencontrées depuis le début de ce travail. La première est celle de ceux ayant un programme parmi les meilleurs du monde. Ces programmes sont commercialisés pour la plupart et leurs auteurs reçoivent des bénéfices impor-

tants. Malheureusement pour la communauté scientifique, la conception de ces programmes reste cachée.

La deuxième catégorie est celle du logiciel libre : les sources sont publiques, tout le monde peut apporter sa contribution à l'édifice, et comprendre la conception de ces programmes.

La troisième catégorie est celle du monde universitaire dans laquelle les chercheurs développent des programmes en publiant leurs idées dans des articles de revues ou de conférences. J'appartiens à cette catégorie et une partie du *travail consiste à publier*, ce qui est fait sur une base régulière. Les publications ne sont jamais évidentes à obtenir. La stratégie suivie a été de commencer en publiant dans des conférences plus faciles d'accès et ainsi de suite jusqu'à des conférences plus renommées et enfin des revues (cf http ://www.math-info.univ-paris5.fr/∼bouzy/publications.html).

La victoire de Deep Blue sur Kasparov en 1997 a sans doute marqué un tournant dans l'histoire de la programmation des échecs, mais elle a aussi facilité les publications des chercheurs sur d'autres jeux que les échecs, tels que le go ou le shogi. En effet, après la victoire du silicium sur la matière grise aux échecs, plusieurs faits importants sont survenus dans le monde académique. Premièrement, l'ICCA (International Computer Chess Association) s'est progressivement transformée en ICGA (International Computer Game Association) [1] afin de s'ouvrir vers d'autres jeux que les échecs. De plus, la conférence biennale "Computers and Games" a été créée en 1998. Pareillement, la conférence quadriennale "Advances in Computer Chess" a été renommée "Advances in Computer Games". Opportunément, les olympiades d'ordinateurs sur les "autres" jeux, interrompues depuis 1991, ont repris en 2000. En conséquence, les publications obtenues ces dernières années sont la combinaison de deux facteurs : d'une part la tournure très positive prise par mon travail, et, d'autre part, le regain académique de la programmation des "autres" jeux depuis quelques années.

2.4 Modèle cognitif ou computationnel ?

Le titre du mémoire de thèse de doctorat était "Modélisation cognitive du joueur de go". Les recherches effectuées depuis sont différentes, mais dans la continuité du travail initial. Donc il est intéressant, neuf ans plus tard, de trouver un titre adéquat au mémoire exprimant à la fois cette continuité et ces différences. Pour cela, il est important de savoir si le modèle conceptuel d'Indigo était computationnel ou cognitif pendant la thèse et ce qu'il est devenu aujourd'hui.

Au cours de la thèse, le programme a été développé avec la volonté de garder sa conception aussi claire que possible. Le programme suivait donc un modèle conceptuel. Comme ce modèle conceptuel était implémenté sur ordinateur, il est évident de dire que ce modèle était computationnel. Cependant, comme le travail de thèse se placait dans le cadre des sciences cognitives, son but était de produire un modèle "cognitif". Pour cela, une correspondance entre les concepts présents dans le programme (les concepts computationnels) avait été établie avec des concepts utilisés par les joueurs humains (les concepts cognitifs). Un objectif

fort était de rester en contact avec la façon de fonctionner des joueurs humains afin de qualifier le modèle d'un point de vue cognitif. De ce point de vue, le modèle était cognitif. Donc en 1995, à la fin de la thèse, le modèle conceptuel était à la fois computationnel et cognitif.

Depuis, les recherches effectuées se placent dans une optique purement informatique ou "computationnelle" : avoir une idée, la spécifier dans un court article, prévoir les résultats attendus, la programmer, lancer les expériences sur ordinateur, récupérer les résultats, les comparer aux résultats attendus et enfin conclure sur sa validité et sur son exploitation future. Même si l'auteur du programme est expert du domaine et inspirateur des idées qui y figurent, le but est de laisser les résultats des expériences sur machine décider de la validité des idées. De ce point de vue, le programme obtenu en 2004 correspond peu à un modèle cognitif et *surtout à un modèle computationnel*. Par ailleurs, Indigo possède encore une grande marge de progression et le modèle correspondant n'est pas terminé. C'est pourquoi ce mémoire d'habilitation est intitulé "Vers un modèle computationnel du joueur de go".

2.5 Objectif

En résumé, l'objectif de ces recherches est de produire un *programme de go*, Indigo, correspondant à un *modèle computationnel* du joueur de go, *aussi bon que possible*. Les indicateurs de résultats sont les progrès réalisés d'une année sur l'autre, les *résultats des compétitions d'ordinateurs* et les *publications* dans des médias reconnus afin de faire profiter la communauté des chercheurs en IA de ces recherches.

2.6 Méthode expérimentale et didactique

Pour atteindre cet objectif, la méthode de chercheur repose sur deux principes. Le premier est expérimental, déjà décrit dans la partie 2.2. Il se résume à avoir en permanence un prototype participant aux compétitions. Ce principe est lourd car faire des expériences a de l'inertie. Le second principe est didactique. Pour trouver, le chercheur doit être à la pointe de son domaine. Le chercheur doit veiller à effectuer des recherches non seulement dans un but de trouver mais aussi dans le but de se former. Par exemple, en 2001, ayant longuement travaillé sur les damiers 19x19 avec l'approche basée sur les connaissances, il était intéressant de se tourner vers les petits damiers dans le but d'y obtenir un premier résultat mais aussi, et peut-être surtout, dans le but de se former à d'autres techniques. Par exemple, travailler sur les petits damiers a permis d'apprendre à utiliser les améliorations de alfa-béta, largement utilisées dans la programmation des autres jeux de réflexion, et de se familiariser en pratique avec la technique l'analyse rétrograde, très utilisée aux échecs et très gourmande en espace disque. Par ailleurs, lorsque les programmes de go jouant sur petits damiers ont été paramétrés, l'utilisation des algorithmes génétiques était guidée par la volonté de se former. De même en 2002, cette démarche a été suivie avec la technique de Monte Carlo. Dans ce cas précis, deux objectifs ont été atteints :

la formation à un domaine nouveau a été réussie et un résultat très positif et effectif de la technique a été obtenu. Cette formation continue aux techniques existantes sera encore poursuivie à l'avenir avec par exemple l'apprentissage par renforcement. Ainsi, à tout moment, même si une découverte n'est pas faite, ce principe de formation continuelle permet au moins de transformer les connaissances du chercheur régulièrement. Cela permet aussi de ne jamais laisser ses recherches sur place et de toujours avancer. Ce qui est essentiel dans un domaine concurrentiel. Quand une technique est utilisée, le but est double : savoir si elle est adaptée à la résolution du problème considéré, et au minimum la maîtriser.

3 Historique

Cette partie donne un aperçu chronologique des recherches et raconte comment, année après année, la méthode de chercheur définie dans la partie 2.6 a été suivie pour atteindre l'objectif défini dans la partie 2.5.

Depuis 1991, les différentes périodes de recherches sont les suivantes : la période de la thèse (1991-1995), la période post-doctorale (1995-1997), deux périodes de recherche (1997-1999 et 2000-2002) comme maître de conférences à l'université Paris 5 dans l'équipe SBC (Systèmes à base de connaissances) dirigée par Dominique Pastre, et enfin l'année 2003 à la fin de laquelle ce document a été commencé.

3.1 La période 1991-1995

Pendant cette période initiale, intéressé par la programmation du jeu de go et l'IA [57], j'ai fait une thèse sur la modélisation cognitive du joueur de go [33] sous la direction de Jacques Pitrat dans l'équipe Métaconnaissances [123] du LAFORIA à l'université Paris 6. L'implémentation de Indigo a permis la validation du modèle cognitif présenté dans la thèse. La première version de Indigo a été implémentée en C++ entre 1991 et 1993. Il y avait quatre niveaux d'abstractions dans Indigo : le niveau "global", le niveau "groupe et territoire", le niveau "formes de base" et le niveau "tactique" [35]. A cette époque, Indigo était une sorte de *système expert complexe*, ne contenant *aucune recherche arborescente globale*. Le système effectuait des recherches arborescentes sur objectifs simples dans le niveau tactique, utilisait des connaissances sous forme de patterns dans le niveau "formes de base", des connaissances sur la vie et la mort des groupes dans le niveau "groupe et territoire". Finalement, dans le niveau global, Indigo donnait une note à chaque intersection reflétant l'urgence à jouer un coup sur cette intersection. Indigo ne vérifiait pas si le coup était effectivement bon ou pas.

3.2 La période 1995-1997

Pendant cette période post-doctorale, le poste d'A.T.E.R. obtenu à l'université Paris 6 a permis de valoriser le travail de thèse par des publications

dans des domaines variés de l'IA : Intelligence Artificielle Distribuée (IAD) [39, 41], sciences cognitives [36], logique floue [32] et représentation de l'incertitude [38], raisonnement incrémental [40], raisonnement contextuel [54]. De plus, cette période a permis de décanter les idées sur la programmation du go poursuivies dans la thèse.

3.3 La période 1997-1999

En 1997, le recrutement comme de maître de conférences à l'université Paris 5 et l'accueil dans l'équipe SBC dirigée par Dominique Pastre a permis la réécriture de la totalité de Indigo. Ce travail a été achevé en 1998. La description de 1995 en quatre niveaux était simplifiée et devenait la fonction d'évaluation de Indigo. Le processus de décision du coup était le suivant : si la position était calme, Indigo effectuait une recherche arborescente à profondeur 1, sinon, il jouait un coup "urgent". L'urgence étant définie par des heuristiques complexes dépendantes du go. Indigo participa à la coupe Ing 1998 et termina 10ème sur 17. Rétrospectivement, ce résultat allait rester le meilleur résultat de Indigo pendant plusieurs années. En 1999, le temps de réponse des fonctions critiques de Indigo a été optimisé avec le but de faire plus de recherche arborescente. L'appel de la fonction d'évaluation à profondeur 1 a été remplacé par de la recherche de quiescence sélective [17] dans laquelle les coups n'étaient engendrés qu'au voisinage du dernier coup et du premier coup de la séquence. Indigo99 obtenait alors 80% de victoires à égalité contre Indigo98. Indigo participa à la coupe Ing 1999 où il termina 13ème sur 16. Un article sur la complexité des sous-jeux du go [42] a valorisé le travail de cette période.

3.4 La période 2000-2002

L'année 2000 En l'an 2000, GNU Go [60], le programme libre de la Free Software Foundation, était déjà devenu assez fort, et surtout il avait dépassé Indigo. Par conséquent, un programme fort était disponible, avec un source modifiable pour faire jouer automatiquement des séries de parties entre Indigo et GNU Go. A partir de l'an 2000, les tests internes devenaient donc plus complets : au lieu de jouer uniquement contre la version de l'année précédente, la version de travail de l'année en cours pouvait aussi jouer contre un programme "externe". Grâce à toutes les parties jouées contre GNU Go-2.4, qui restait meilleur que Indigo (deux pierres plus fort), Indigo2000 fut alors 4 pierres plus fort que Indigo99. Indigo participa aux olympiades d'ordinateurs à Londres en août 2000 et finit 5ème sur 6. Dans le compte-rendu de cette compétition [169], Nick Wedd, organisateur et arbitre, décrit un symptôme caractéristique de l'état actuel de la programmation de go et de sa difficulté. A la fin d'une des deux parties jouées contre Aya [174], Indigo et Aya ont passé tous les deux alors que, selon les spectateurs de la partie, un groupe était vivant si Indigo jouait en premier, et mort si Aya jouait en premier. Aucun des deux programmes ne le voyait et pour les spectateurs l'issue de la partie en dépendait. Le problème de l'arbitre était alors de savoir qui était le gagnant. Entre joueurs humains, ce cas ne se produit

généralement pas car, en cas de désaccord, les joueurs se parlent et continuent la partie jusqu'à trouver un accord. Pour les joueurs artificiels, le protocole de fin de partie en cas de désaccord n'existe pas et c'est à l'arbitre du tournoi de décider. En fait, étant donné que Indigo interprétait le groupe crucial comme mort, j'ai honnêtement abandonné cette partie, ce qui a soulagé l'arbitre. Dans le cas contraire, l'arbitre aurait eu un petit souci.

L'année 2001 L'année 2001 a marqué le début de l'application de la méthode didactique décrite dans la partie 2.6. La recherche arborescente et l'analyse rétrograde ayant montré leur succès sur d'autres jeux (échecs et othello), elles ont été utilisées et testées sur des damiers de très petite taille, inférieure à 4x4. Le programme Gic-Gac-Goe, d'analyse rétrograde appliquée au go 4x4 [44] et un programme de recherche arborescente utilisant alpha-béta et ses améliorations [43] ont été développés dans l'objectif double de découverte et de formation. Ces deux développements ont permis de mieux comprendre le domaine de la programmation des échecs. En parallèle de cette formation, un article sur le raisonnement spatial au go a été publié [45]. De plus, le long article décrivant l'état de l'art orienté IA de la programmation du go, écrit avec Tristan Cazenave pour la revue Artificial Intelligence, commencé deux ans avant, était terminé [55].

L'année 2002 En 2002, le temps de réponse du programme était suffisamment optimisé pour que des expériences sur la vie et la mort des groupes, non complètement encerclés, soient possibles. Sur une base de test constituée de problèmes de vie et mort (rencontrés en cours de parties réelles et incorrectement traités par Indigo), Indigo avec le résolveur de vie et mort résolvait ces problèmes mais, sur la base de confrontations sur des parties complètes, Indigo avec ce résolveur ne jouait pas mieux que Indigo sans le résolveur. L'intégration d'un module performant sur une sous-classe de problèmes avec le reste du programme échouait sur les parties complètes. Le module de recherche arborescente sur la vie et la mort n'a donc pas été intégré dans Indigo2002. Indigo2002 joue rapidement : en deux minutes par partie sur un 450 Mhz. Son exécutable est accessible en ligne et son processus de décision est décrit dans [51]. En juillet 2002, cette version a participé à la seconde "coupe du 21ème siècle" [5] à Edmonton au Canada et a fini 10ème sur 14 [49]. En novembre, elle a terminé 6ème sur 10 au festival de la programmation du go organisé à Guyang en Chine. En 2002, un article étudiant expérimentalement les fonctions d'évaluation basées sur des principes de distance et de dimension [47] et un autre décrivant l'application de la morphologie mathématique à la programmation du go [50] ont été publiés.

3.5 L'année 2003

L'année 2003 a vu apparaître les premiers résultats d'expériences débutées en septembre 2002 sur Monte Carlo. En 2001, Bernard Helmstetter, doctorant de Tristan Cazenave à l'université Paris 8, avait écrit un programme de Monte

Carlo go nommé Oleg reproduisant l'expérience de Brügmann [59]. Par ses premiers résultats sur damiers 9x9, Bernard Helmstetter m'avait convaincu de l'urgence d'étudier une telle approche. Plutôt que de refaire exactement l'expérience de Brügmann qui est un modèle de recuit simulé, c'est l'expérience d'Abramson qui a été refaite [7]. Elle consiste en un modèle de fonction d'évaluation basé sur des statistiques. Donc en 2003, Bernard Helmstetter et moi avions chacun un programme, Oleg et Olga, basé sur l'approche Monte Carlo. Ces deux programmes jouaient à égalité contre Indigo2002 sur 9x9, alors qu'ils ne contenaient que des connaissances sur les yeux, et rien d'autre, et que Indigo contient une multitude de connaissances du le go. C'était vraiment très étonnant [56] et surtout très motivant pour développer plus loin cette approche. Disposant déjà d'une base de connaissances importantes dans Indigo2002, une nouvelle version de Indigo intégrant les avantages de Monte Carlo et des connaissances a été construite. L'idée de cette construction était double. Premièrement, utiliser des connaissances rudimentaires au sein des parties aléatoires, pour que celles-ci deviennent pseudo-aléatoires et fournissent des moyennes plus significatives que celles calculées avec des parties aléatoires utilisant la probabilité uniforme. Deuxièmement, utiliser des connaissances et des recherches locales tactiques pour filtrer les coups en entrée du module Monte Carlo et accélérer le temps de réponse du programme. Cette construction bipolaire entre connaissances du domaine et statistiques a vraiment donné de très bons résultats. Elle a été publiée [48] et elle a suscité l'attention d'autres programmeurs lors des dernières olympiades d'ordinateurs à Graz auxquelles Indigo a participé [6]. Indigo a terminé 4ème sur 10 participants sur damiers 9x9 [163] et 5ème sur 11 participants sur damiers 19x19 [78]. Ce résultat est très satisfaisant car Indigo termine dans la première moitié d'une compétition internationale pour la première fois. Ce bon résultat devrait être confirmé dans les prochaines compétitions. La conclusion actuelle est que l'approche de Monte Carlo vaut la peine d'être travaillée encore.

4 Publications

Cette partie commente des publications majeures de la recherche effectuée et rassemble la liste exhaustive des publications depuis 1992. Les publications citées dans cette partie sont pour la plupart accessibles par le web (http ://www.math-info.univ-paris5.fr/ bouzy/publications.html). En particulier, les publications majeures le sont toutes sous forme de pré-prints.

4.1 Publications majeures

Cette partie cite cinq publications majeures reflétant et valorisant au mieux le travail effectué. Les deux aspects, reflet et valorisation, vont de paire. En effet, pour un travail donné, l'effort de publication a été poursuivi depuis la publication dans un colloque ou congrès national vers une publication dans une revue nationale ou internationale lorsque cela a été possible, en passant par une publication dans des actes de conférences internationales. Par exemple,

l'approche Monte Carlo associée à des connaissances a d'abord été publiée dans les actes de la conférence JCIS [48] en 4 pages, puis dans la revue Information Sciences en 10 pages [52]. L'application de la morphologie mathématique a été publiée d'abord dans une conférence nationale en 1995 [32] puis, après l'intérêt suscité par ce travail dans le domaine de la programmation du go, en 2003 dans une revue internationale [50]. Le travail sur le raisonnement spatial au go a vécu de nombreuses versions : celle d'un workshop associé à l'ECAI en 1996 [37], celle des Journées Nationales des Modèles de raisonnement du GdR I3 en 1999 [4] avant d'aboutir à celle de la Revue d'Intelligence Artificielle [45] en 2001. En collaboration avec Tristan Cazenave, la publication de [55], état de l'art d'un domaine en 70 pages, a été en elle-même un travail de longue haleine. L'article [51] décrivant le processus de décision utilisé par Indigo en 2002 a été publié dans le journal de l'association internationale de la programmation des jeux. C'est pourquoi les cinq publications reflétant au mieux ce travail de recherche figurent dans des revues internationales ou nationales.

[52], "Associating domain-dependent knowledge and Monte Carlo approaches within a go program" , 10 pages, à paraître dans Informations Sciences, (2004).

Cet article montre comment l'approche Monte Carlo a été associée avec une approche basée sur des connaissances dans le programme Indigo. Même si cette publication relate un travail effectué en 2003, c'est la publication qui reflète le mieux l'ensemble du travail effectué depuis 1997 du point de vue de son résultat. En effet, l'approche basée sur des connaissances, suivie entre 1997 et 2002, n'avait pas permis à elle seule d'obtenir des résultats vraiment concluants. L'approche Monte Carlo commencée en 2002 est prometteuse et donne de bons résultats couplée avec l'approche basée du des connaissances. Donc, sur l'ensemble de la période 1997-2003, je considère cette association de deux approches complémentaires comme une réussite technique. Cet article a d'abord été publié en version courte (4 pages) à la conférence JCIS.

[50] "Mathematical morphology applied to computer go" , International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI), vol. 17 n2, pp. 257-268, (2003).

Cet article montre comment la morphologie mathématique est appliquée pour reconnaître les territoires et l'influence au go. Il décrit des opérateurs ressemblant à l'érosion et la dilatation, adapté dans le cas du go. Ce travail a été ré-utilisé dans le programme libre GNU Go et dans Indigo. Le modèle décrit par cet article avait été trouvé pendant la thèse avant 1995. L'implémentation décrite par cet article a été faite entre 1997 et 2000 lors des optimisations du temps de réponse de Indigo. C'est donc un article reflétant un aspect du travail effectué sur le long terme.

[55] "Computer Go : an AI oriented Survey" , avec T. Cazenave, Artificial Intelligence Journal, vol. 132, n1, pp. 39-103, (2001).

Cet article présente un état de l'art de la programmation du go suivant le point de vue de l'intelligence artificielle. Il est désormais cité dans de nombreux articles sur la programmation du go. Pour certains aspects détaillés, il prend pour base les travaux effectués par Tristan Cazenave et moi-même dans ce domaine avec Indigo et GoLois. Pour les autres aspects, il relate les techniques variées d'intelligence artificielle utilisées par l'ensemble de la communauté des programmeurs de go. Ainsi, il permet aux chercheurs en IA de mieux connaître la problèmatique de la programmation du go et ses solutions actuelles.

[45] "Les concepts spatiaux dans la programmation du go" , Revue d'Intelligence Artificielle, vol. 15, n2, (2001), pp.143-172.

Cet article est une illustration des techniques de raisonnement spatial. Il présente la fonction d'évaluation d'un programme de go utilisant de nombreux concepts spatiaux. Les concepts spatiaux sont la connexion, l'encerclement, le territoire, l'influence. Ils sont reconnus avec des outils tels que la morphologie mathématique, la distance de Hausdorff.

[51] "The move-decision strategy of Indigo" , International Computer Game Association Journal (ICGAJ), vol. 26, n1, pp.14-27, (2003).

Cet article décrit le processus de décision utilisé par Indigo en 2002. Le processus de décision utilise plusieurs méthodes de choix de coups. La méthode (calme(effectue de la recherche de quiescence globale sélective. La méthode (urgente(sélectionne les coups urgents avec une base de connaissances. Ces coups sont ensuite vérifiés par la recherche de quiescence globale sélective. D'autres méthodes sont étudiées : la recherche monocolore, la recherche arborescente sur la vie et la mort des groupes.

4.2 Liste exhaustive des publications

Cette partie dresse la liste exhaustive des publications en distinguant les publications dans des revues internationales ou nationales, des conférences internationales ou nationales, avec ou sans comité de lecture.

4 Revues internationales avec comité de lecture : [50, 51, 52, 55]

2 Revues nationales avec comité de lecture : [45, 49]

7 Conférences internationales avec comité de lecture : [38, 41, 47, 48, 53, 54, 56]

6 Workshops internationaux avec comité de lecture : [35, 34, 37, 40, 42, 44]

5 Conférences nationales avec comité de lecture : [30, 31, 32, 36, 39]

2 Revues sans comité de programme : [15, 57]

5 Présentation thématique

La programmation du jeu de go est un domaine de prédilection pour faire des expériences en IA [55, 57]. La communauté de l'IA est composée de multiples domaines, ou thèmes, et il est intéressant de faire le point entre l'interaction ces

thèmes et les recherches effectuées. D'abord, la sous-partie 5.1 présente l'interaction avec deux domaines qui ont interagit fortement avec les recherches menées pendant la thèse et pendant la période post-doctorale : les sciences cognitives et l'IAD. Ensuite, les sous-parties suivantes présentent les domaines qui ont été en interaction forte et qui sont toujours utilisés dans le programme actuel : la gestion de l'incertitude et la théorie des jeux sommables (sous-partie 5.2), l'analyse rétrograde et la recherche arborescente (sous-partie 5.3), le raisonnement spatial et les fonctions d'évaluation (sous-partie 5.4). L'utilisation de l'approche Monte Carlo est montré dans la sous-partie 5.5. Enfin, le domaine de l'apprentissage par renforcement [150] ne sera pas mentionné dans cette partie car il sera développé dans la partie 8.

Les sous-parties de cette partie sont de tailles et natures différentes. Cela reflète mon intérêt actuel sur chacun de ces domaines. L'intérêt porté à un domaine donné peut correspondre à un état de l'art poussé, par exemple dans la sous-partie traitant de la recherche arborescente, ou bien correspondre à une utilité effective dans le cadre de la fabrication d'un programme de go, par exemple dans la sous-partie sur Monte Carlo.

5.1 Sciences Cognitives et IAD

Le travail de thèse s'inscrivait dans le cadre des sciences cognitives [33]. Comme déjà expliqué dans la partie 2.4, le modèle conceptuel de Indigo était à la fois un modèle computationnel (au sens "implémenté dans un programme tournant sur ordinateur") et un modèle cognitif (au sens "inspiré du comportement humain"). Dans le travail de thèse, le modèle computationnel de Indigo a été mis en correspondance avec le modèle cognitif. L'hypothèse de travail faisait correspondre un concept computationnel à un concept cognitif humain, conscient ou non. Lorsque les verbalisations de joueurs de go contenaient des termes se référant à ces concepts computationnels, ceux-ci étaient déclarés "conscients" et les autres, "non conscients" ou "implicites" [36]. Cependant, désormais, plus le travail avance, plus il devient computationnel.

Une position de go est décomposable en sous-positions interagissant fortement les unes avec les autres, notamment lorsque deux groupes voisins ennemis combattent chacun pour tuer l'autre. Dans un tel combat, le groupe qui a le plus de libertés ou le plus d'yeux gagne le combat. D'un point de vue conceptuel, il est pratique d'avoir un objet "interaction" qui rassemble les propriétés relatives à l'interaction entre ces deux groupes : le nombre d'yeux, de libertés communes ou libertés propres des deux groupes et le gagnant du combat. Les articles [39, 41] font correspondre un "groupe", au sens du go, à un "agent", au sens de l'IAD. Ainsi, l'exemple du go confirme la place tenue par l'interaction en IAD. Cependant, la correspondance groupe-agent n'est que formelle, la force actuelle de Indigo, toute relative, résulte, entre autres choses, de la gestion des interactions ennemies entre les groupes et peu de cette correspondance.

5.2 Jeux sommables et gestion de l'incertitude

En observant qu'une position de go est complexe mais structurée, on peut tenter de décomposer la position en sous-positions, puis étudier séparément les sous-positions et enfin calculer l'évaluation de la position entière à partir des évaluations des sous-positions. Si les sous-positions sont indépendantes, on peut utiliser la théorie des jeux sommables de Conway [81, 22] et décomposer le jeu global en sous-jeux. Avec l'hypothèse d'indépendance entre les sous-jeux, de nombreux travaux théoriques ont été faits, dont le célèbre "mathematical go" de Berlekamp [19, 23] ou "decomposition search" de Martin Müller [115], suite de sa thèse [114].

En fait, les sous-positions du go sont très dépendantes les unes des autres, c'est une évidence de l'écrire. Malgré tout, même si c'est incorrect en théorie, décomposer la position en sous-positions diminue la complexité du problème et permet d'amorcer l'étude. En pratique, on peut décomposer la position en sous-positions en reconnaissant les groupes et les territoires. Cela se fait avec des connaissances du domaine [33]. Ensuite, il faut décrire les sous-positions (les groupes) de manière utile, puis recomposer la fonction d'évaluation avec ces descriptions locales. J'ai donc travaillé sur la description des sous-jeux [42] et sur la recomposition de la fonction d'évaluation [38].

5.2.1 Fonction d'évaluation incluant une incertitude

L'article [38] a été écrit lorsque Indigo était un système expert notant les coups avec des règles et jouant simplement le coup ayant la meilleure note. Indigo faisait beaucoup d'erreurs liées à l'absence de vérification de ses propres coups. En revanche, il constatait ses erreurs au coup d'après. Donc, beaucoup de ces erreurs pouvaient être supprimées en utilisant simplement une recherche arborescente à profondeur un, appelant une fonction d'évaluation. Une fonction d'évaluation correspondant au générateur de coups qu'était Indigo a été conçue. Le problème crucial était celui des combats entre les groupes. Selon l'issue d'un combat, l'appartenance des intersections à l'un ou à l'autre des joueurs change et la valeur de l'évaluation également. Il fallait donc que la fonction d'évaluation inclue cette incertitude sur l'issue du combat entre les groupes. Dans cet article [38], pour chaque groupe "incertain", c'est-à-dire ni vivant ni mort, une incertitude est définie par la différence entre l'évaluation si on suppose le groupe vivant et l'évaluation si on suppose le groupe mort. Quand deux groupes combattent, plus l'issue est proche, plus les groupes sont gros et plus l'imprévisibilité augmente. La fonction d'évaluation présentée dans cet article inclue donc une incertitude. Celle-ci est toujours utilisée dans le programme actuel.

5.2.2 Mesurer la force des groupes avec les nombres P et Q

L'article [42] étudie la pratique de la décomposition du jeu global en jeux de la vie et de la mort des groupes et définit l'importance de la "force" des groupes par des nombres P et Q. Dans le jeu du groupe, la fonction d'évaluation prend trois valeurs : MORT, VIVANT, AUTRE. Une recherche arborescente sur la vie

et la mort d'un groupe permet en pratique de raffiner l'état AUTRE en trois sous-états : "mort", "vivant", "autre". Un groupe est mort (respectivement vivant) s'il est AUTRE et si la recherche arborescente permet de conclure à sa mort (respectivement vie). S'il est AUTRE mais ni mort et ni vivant, alors il est autre. Avec une recherche arborescente de plus en plus complète le nombre de groupes dans l'état autre diminue, le nombre de groupes dans l'état vivant et le nombre de groupes dans l'état mort augmentent. Malheureusement, en pratique, la recherche est limitée et l'état autre reste grand.

La recherche arborescente peut être faite en deux temps : une première dans laquelle l'ami du groupe joue en premier et une seconde dans laquelle l'ennemi joue en premier [81]. Ce que l'on notera $\{ami|ennemi\}$. Dans ces conditions, il y a 9 résultats possibles. En faisant l'hypothèse que jouer doit avoir un effet strictement positif, ces 9 résultats se réduisent à : $\{autre|mort\}$, $\{vivant|mort\}$, $\{vivant|autre\}$. Mais, en faisant cela, on élimine le cas $\{autre|autre\}$ très important et fréquent en pratique. C'est le cas d'un groupe dont la recherche arborescente ne conclut ni sur sa mort ni sur sa vie, que ami ou ennemi joue en premier. En pratique, ce cas doit être traité par un programme de go.

De deux choses l'une, soit la fonction d'évaluation traite ce cas avec des règles statiques disant si le groupe est fort ou faible, attaquable ou pas, etc. Soit elle ne le fait pas et il peut alors être utile de faire de la *recherche mono-couleur* en supposant que le même joueur joue plusieurs coups de suite pour faire vivre un groupe ami ou tuer un groupe ennemi. Dans [42], des nombres mesurant la force et la faiblesse d'un groupe on été définis. P désigne le nombre de coups ennemis successifs pour tuer un groupe et Q le nombre de coups amis successifs pour faire vivre le groupe. L'avantage de cette définition est son indépendance des connaissances sur le go. C'est une recherche arborescente mono-couleur qui détermine P et Q. En pratique, P et Q peuvent être grands. Pour cela, il suffit que la position soit trop complexe pour les recherches arborescentes bi-couleur sur la vie et la mort des groupes. En théorie combinatoire des jeux [81], avec des recherches arborescentes bi-couleur supposées complètes, les seuls états possibles sont $\{vivant|vivant\}$, $\{vivant|mort\}$, $\{mort|mort\}$, $\{mort|vivant\}$. En considérant que jouer a un effet strictement positif, il ne reste plus que l'état $\{vivant|mort\}$ et on constate que $P + Q <= 1$ en théorie. Dans [42], cette classification sur la vie et la mort des groupes a été comparée aux états GI, GP, IP, IIP, etc. de jeux de connexion ou capture de chaîne définis par Tristan Cazenave [70] et au "Possible Omission Number" (PON) de Tajima et Sanechika [151].

5.3 Recherche arborescente

Cette sous-partie présente l'état de l'art en recherche arborescente et l'apport des recherches effectuées à ce domaine. Elle est la plus longue de la partie 5 car la recherche arborescente est le paradigme dominant de la programmation des jeux de réflexion. Le but premier de cette sous-partie est donc la présentation de l'état de l'art. La relation entre la recherche arborescente et la programmation du go est jusqu'à maintenant très spéciale : contrairement aux autres jeux de réflexion dans lesquels la recherche arborescente est globale, les recherches arborescentes

effectuées par les programmes de go sont pour la plupart locales et très peu globales. Néanmoins, certains programmes de go, dont Indigo, font un peu de recherche globale, et, la puissance des machines augmentant régulièrement, ils en feront de plus en plus. Ainsi la recherche arborescente globale va devenir un domaine de plus en plus utile à la programmation du go.

5.3.1 Travaux existants

Il existe de nombreux articles étudiant alfa-béta en pratique : celles de Marsland [66, 109] ou en théorie : celles de Knuth [99] ou celle de Pearl [122]. Un état de l'art assez récent est celui de Junghanns [96]. Schaeffer présente une tentative intéressante d'unification de la recherche mono-agent et de celle avec deux joueurs [138]. La classification des publications sur alfa-béta est la suivante : alfa-béta à profondeur fixée et ses "améliorations", algorithmes alfa-béta de type "meilleur en premier", algorithmes incluant des statistiques ou probabilités, et les applications de alfa-béta pour le go.

Améliorations de alfa-béta à profondeur fixée. alfa-béta [109] offre l'avantage d'être un algorithme peut coûteux en mémoire. En effet, à un instant donné, une seule séquence est explicitement en mémoire. L'algorithme utilise donc un espace mémoire linéaire en fonction de la profondeur de recherche. En revanche, il utilise un temps exponentiel en fonction de la profondeur et, sans ses "améliorations", il a de piètres performances.

Tables transpositions et "Iterative deepening". La première amélioration est l'utilisation d'une table de transpositions [95]. Pour ne pas effectuer une recherche partant d'une position déjà rencontrée antérieurement, premièrement l'algorithme stocke le résultat de chaque recherche partant d'une position dans une table indexée par les positions, appelée table de transpositions. Deuxièmement, avant de démarrer une recherche issue d'une position donnée, il interroge cette table et utilise le résultat si la table le fournit. C'est une amélioration obligatoire. La seconde amélioration obligatoire est "iterative deepening" [145, 100]. Elle consiste à faire une recherche à profondeur 1, puis à profondeur 2, etc jusqu'à la profondeur fixée ou jusqu'à l'épuisement du temps de calcul. Intuitivement, cette idée ne parait pas efficace : puisque l'on fait une recherche à profondeur N après avoir fait une recherche à profondeur $N-1$, un traitement semble être répété. En fait, plusieurs remarques permettent de comprendre pourquoi cette amélioration est efficace. Premièrement, elle est combinée avec les tables de transpositions. L'algorithme écrit non seulement la valeur de la position dans la table de transpositions mais aussi le "meilleur" coup trouvé. Lors de l'itération suivante, ce "meilleur" coup est utilisé en premier. alfa-béta étant très sensible à l'ordre des coups, ce "meilleur coup" de la profondeur $N-1$ accélère en général la recherche effectuée à profondeur N et le traitement semblant être répété ne l'est pas. Deuxièmement, le nombre de nœuds parcourus à profondeur $N-1$ est significativement plus petit que celui de la profondeur N (d'un facteur égal au facteur de branchement de l'arbre du jeu), donc le surplus

de recherche n'est pas très important. Enfin, "iterative deepening" trouve la séquence solution la plus courte, ce que ne fait pas alfa-béta tout seul. alfa-béta sans "iterative deepening" peut étudier des séquences très longues sans résultat alors qu'une séquence solution plus courte existe. Ces avantages rendent iterative deepening "obligatoire".

"Null move", "History Heuristic". Par ailleurs, on peut utiliser l'"history heuristic" [133] ou l'heuristique du "killer move" pour un meilleur ordre des coups. Une autre heuristique très intéressante est celle du "null move" [83]. Elle consiste à commencer une recherche à une profondeur donnée par une recherche un peu moins profonde (donc beaucoup moins coûteuse en temps) et commençant par un coup "passe". Ainsi, l'algorithme effectue d'abord une recherche dans laquelle l'adversaire joue deux coups de suite. Cela donne une première approximation d'une borne de la recherche normale que l'algorithme va faire ensuite. C'est une amélioration efficace utilisée dans la plupart des programmes d'échecs.

Recherche de quiescence, extensions singulières Lorsque les positions cherchées sont instables, il est maladroit d'appeler la fonction d'évaluation sur ces positions. Au lieu de cela, l'algorithme doit appeler une recherche très sélective dans laquelle seuls les coups "urgents" sont sélectionnés. Cette recherche sélective [17] aboutit à des positions "calmes", sans coup urgent, évaluables avec un niveau de confiance élevé. La difficulté de cette approche réside dans la définition d'une position "calme" et des coups "urgents". Aux échecs, un coup "urgent" pourra être une réponse à un échec au roi, un échec au roi, une capture de pièce, etc. Au go, un coup urgent pourra être la capture d'une chaîne, une connexion, etc. Cette approche rencontre alors dans le problème difficile de la définition des connaissances dépendantes du domaine.

Selon que le meilleur coup d'un nœud de l'arbre est clairement devant les autres, on peut utiliser un mécanisme de crédit permettant de développer des extensions singulières [12], mais on sort du cadre de l'alfa-béta à profondeur fixée.

"SCOUT" et "MTDF". Une autre idée transversale à toutes les précédentes est l'idée déjà ancienne du "test" de l'algorithme SCOUT [121]. Cela consiste à faire confiance à l'ordonnancement des coups proposés par les connaissances sur le domaine et de considérer que le premier coup va être le bon. Pour le deuxième coup et les suivants, on ne fait pas une vraie recherche mais seulement la vérification qu'il n'est pas meilleur que le premier en restreignant la largeur de la fenêtre alfa-béta. La vérification coûte moins cher en temps que la recherche normale. Donc dans le bon cas, on gagne du temps. Si la vérification est fausse, le mauvais cas, alors on lance une recherche normale. L'un dans l'autre, SCOUT est une très bonne amélioration.

Le meilleur algorithme actuel de type alfa-béta en profondeur d'abord à profondeur fixée est MTDF [124]. MTDF est utilisé dans beaucoup de pro-

grammes d'échecs actuels. Cet algorithme utilise "iterative deepening" et une table de transpositions et il effectue des vérifications au sens de [121] avec des fenêtres alfa-béta de taille minimale, le grain de la fonction d'évaluation. Chaque vérification se termine soit par un "fail-high" et la valeur minimax est démontrée supérieure à la fenêtre, soit par un "fail-low", le cas opposé. Comparées à une recherche normale sans fenêtre, les vérifications sont en général rapides. MTDF est en fait une classe d'algorithmes utilisant alfa-béta.

Algorithmes "meilleur en premier". D'autres algorithmes ne sont pas en profondeur d'abord et méritent une attention particulière. Leur avantage est d'explorer moins de nœuds que les algorithmes en profondeur d'abord. Leur inconvénient est l'obligation de garder explicitement tout l'arbre exploré en mémoire. Ils sont donc exponentiels en espace mémoire. Comparés aux algorithmes en profondeur d'abord, ils passent plus de temps à parcourir l'arbre en mémoire pour savoir quel est le prochain nœud à développer.

SSS* de Stockman [148] est un algorithme astucieux montrant comment associer deux idées très différentes, le min-max et A*. En définissant et utilisant le concept d'"arbre solution", [148] montre que SSS* est un algorithme min-max "supérieur" à alfa-béta au sens où il explore moins de nœuds que alfa-béta. En fait, depuis l'apparition de MTDF [124], Aske Plaat a montré premièrement que SSS* est une instance particulière de la classe d'algorithmes MTDF, et deuxièmement que ce n'est pas la meilleure. Par là, Aske Plaat a montré que, contrairement à ce que [148] défendait, un algorithme min-max n'est pas meilleur que alfa-béta. En fait ces deux publications, [124] et [148], ne sont pas contradictoires, car Stockman considérait alfa-béta *sans* ses améliorations alors que MTDF utilise alfa-béta *avec* les tables de transpositions.

Proof-number search de Victor Allis [11] est un autre algorithme min-max dans lequel des nombres sont associés à chaque nœud et mesurent la capacité de ce nœud à prouver la valeur minimax de la racine. Ainsi, l'algorithme expand toujours les nœuds les plus "prouvants". Hans Berliner est l'auteur de B* [24], algorithme dans lequel la fonction d'évaluation doit avoir deux valeurs : l'une pessimiste et l'autre optimiste. B* oriente sa recherche afin de prouver que la valeur minimax pessimiste du meilleur nœud situé à profondeur 1 est supérieure à toutes les valeurs optimistes des autres nœuds situés à profondeur 1. Korf et Chickering ont publié "Best-First" [101], un algorithme qui expand toujours le nœud terminal situé sur la variante principale. Cet algorithme ne peut marcher convenablement que si la fonction d'évaluation, appelée sur tous les nœuds, est fiable. La recherche utilisant les nombres conspirants de McAllester [111] et Schaeffer [134] analyse quels sont les nœuds terminaux susceptibles de changer la valeur de la racine si on les développe.

Utilisation des statistiques ou d'une règle de back-up différente du min-max. La règle de "back-up" est la règle permettant de calculer la valeur d'un nœud père en fonction des valeurs des nœuds fils. Dans tout ce qui a précédé la règle de back-up est celle du mini-max. Rivest présente une formule

de back-up basée sur des racines et des puissances unifiant la formule de back-up Monte Carlo basée sur la moyenne et la formule de back-up classique du min-max [129]. L'idée est intéressante théoriquement mais, à ma connaissance, elle n'a pas donné de résultats en pratique. Par ailleurs, Baum et Smith [16] ou Palay [120] font l'hypothèse que la fonction d'évaluation n'est pas réduite à une valeur mais est une distribution de valeurs. La règle de back-up est alors une règle s'appliquant sur des distributions de probabilités. Cette classe d'algorithmes basée sur des probabilités est assez peu développée en regard du reste. Elle pourra être développée en relation avec l'approche Monte Carlo (cf 5.5) dans laquelle la fonction d'évaluation est une moyenne d'un échantillon de valeurs, lui-même interprétable comme une approximation d'une distribution de probabilités.

ProbCut [62] de Michael Buro a contribué au succès de Logistello, le meilleur programme d'othello. ProbCut effectue une recherche à profondeur faible pour obtenir une approximation des bornes de la fenêtre alfa-béta. ProbCut fait l'hypothèse que ces valeurs sont corrélées avec celles de la fenêtre alfa-béta normale, cela lui permet d'élaguer certains coups avec un niveau de confiance suffisant et un coût inférieur à celui d'une recherche normale. La fonction d'évaluation de Logistello a été construite par une régression linéaire [61]. L'article de Sadikov, Kononenko et Bratko [130] présente une étude pratique très intéressante sur les finales KRK (roi+tour contre roi) aux échecs pour cerner le lien entre recherche arborescente et erreur dans les fonctions d'évaluations. Il est prouvé sur l'exemple des finales KRK que la recherche arborescente appliquée à des fonctions d'évaluations erronées introduit un biais dans les valeurs minimax mais que cela ne change pas le niveau de jeu.

Algorithmes dérivés de alfa-béta et appliqués au go. Au go, Chen a étudié l'impact d'une erreur dans la fonction d'évaluation et celui de la sélectivité de la génération de coups sur les valeurs minimax de la racine et le niveau du programme [77]. Récemment des algorithmes basés sur alpha-beta ont été inventés et appliqués au sous-jeux du go, le jeu de la capture de chaîne en particulier. "Lambda-search" [156] de Thomas Thomsen, "Abstract Proof Search" [71], GAPS [73] et [72] de Tristan Cazenave sont très efficaces pour résoudre les jeux avec des menaces. Le go 4x4 a été résolu en 2000 [141] par Sei et Kawashima, le 5x5 en novembre 2002 [164] par Erik van der Werf. L'expérience du go 2x2, 3x3 et 4x4 [43] a faite. Ken Chen a aussi étudié la possibilité de faire de la recherche arborescente globale au go [75]. "Decomposition Search" [115] de Martin Müller utilise la décomposition du jeu global en sous-jeux locaux [116]. Il a été appliqué avec un succès relatif à certaines positions.

5.3.2 Recherche arborescente dans Indigo

Cette partie présente le travail effectué en regard de la recherche arborescente : les divers types de recherches arborescentes qui ont existé dans Indigo jusqu'en 2002 [51], l'analyse rétrograde [44], la recherche arborescente sans connaissance sur des petits damiers [43] et enfin l'association d'une recherche sélective

globale avec Monte Carlo.

Les divers types de recherches arborescentes avant 2002 Jusqu'en 2002, Indigo utilisait de nombreuses techniques de recherche arborescente [51]. Au niveau tactique sur les shichos[2], moyennant un bon ordonnancement des coups les recherches arborescentes sont linéaires la plupart du temps, la recherche peut être exhaustive et correcte [35]. Au niveau de la vie et la mort des groupes, on doit sélectionner les coups avec des heuristiques sur le go. Quand un résolveur de problèmes de vie et mort sur des groupes non complètement encerclés a été développé, il fallait également limiter la profondeur de recherche. Donc la sélectivité plus la profondeur fixée rendaient l'algorithme assez peu performant. Au niveau global, Indigo2002 utilisait de la recherche à profondeur 1 avec ensuite de la recherche de quiescence et une sélectivité très forte interdisant les coups dont le lieu était trop loin du lieu du premier coup et du lieu du dernier coup de la séquence en cours. Le problème majeur de la recherche arborescente à profondeur fixée sur des positions non terminales est souvent l'inadéquation de la fonction d'évaluation appelée sur ces positions. Tant que la recherche arborescente appelle une fonction d'évaluation contenant des erreurs, cette recherche arborescente ne peut aboutir à des conclusions fiables. Les valeurs minimax contiennent les mêmes erreurs que celles de la fonction d'évaluation.

Recherche arborescente et analyse rétrograde sur petits damiers En 2001, l'analyse rétrograde [157, 158], utilisée avec succès aux échecs, a été appliquée à des petits patterns du go. L'algorithme démarre avec des patterns NxN ($N <= 4$) terminaux pour la fonction d'évaluation de base du go. A partir des patterns existants, il crée des nouveaux patterns en déjouant les coups et il les qualifie avec des switches $\{Noir|Blanc\}$ selon que noir ou blanc joue le premier dans le pattern. L'algorithme recommence cette procédure tant que tous les patterns ne sont pas tous créés et qualifiés. Cette technique est intéressante dans le cadre de l'acquisition automatique de patterns [44].

En 2001, l'expérience de Sinichi Sei sur la résolution du go 4x4 [141] a été reproduite en utilisant un alfa-béta avec les améliorations classiques [43] : transpositions [95], iterative deepening [145, 100], history heuristic [133], null move [83], MTDF [124].

La recherche arborescente associée à Monte Carlo en 2003 En 2003, Indigo est basé sur trois modules, un module de génération de coups et d'évaluation de positions utilisant des connaissances du go, un module Monte Carlo contenant un moteur de parties aléatoires, et un module de recherche arborescente contenant min-max global à profondeur 1, 2 ou 3 selon la taille du damier décrit ici brièvement. A chaque expansion de nœud, le générateur de coups basé sur les connaissances sélectionne w_+ coups pour le module Monte Carlo ($w_+ = 7$). Si la profondeur du min-max est 1, le module Monte Carlo

[2] séquence de coups tactiques visant à capturer une chaîne de pierres

calcule une moyenne pour chaque nœud terminal et fournit le coup avec la meilleure moyenne. Si la profondeur du min-max est 2 ou 3, Monte Carlo calcule itérativement les moyennes pour les nœuds à profondeur 1, 2 puis 3. Si le nœud est non terminal, les moyennes calculées à la profondeur suivante servent à sélectionner w_- coups ($w_- = 3$) sur les w_+ coups initiaux. Lorsque les w_- coups ont été sélectionnés, on expand les w_- nœuds fils. Si un nœud est terminal, le Monte Carlo calcule sa moyenne. A chaque nœud non terminal, "progressive pruning" est utilisé pour réduire le nombre de nœuds fils de w_+ à w_-. Lorsque Monte Carlo calcule des moyennes à une profondeur, les valeurs mini-max des nœuds ancêtres sont mises à jour avec la règle de back-up classique et la règle du "progressive pruning" est ré-appliquée partout. Ainsi, au fur et à mesure des parties aléatoires jouées, le nombre de nœuds diminue. La recherche arborescente s'arrête lorsqu'un seul coup est possible à la racine, ou lorsque qu'un nombre d'itérations maximal est atteint ou lorsque tous les coups possibles ont une valeur égale. Cet algorithme est sensible au test de supériorité d'un coup sur l'autre ($R = 1.7$), au test d'égalité entre plusieurs coups ($\sigma = 0.4$), aux nombres w_+ et w_- et à la profondeur maximale. Ce mécanisme est décrit dans [53].

5.4 Fonctions d'évaluation et raisonnement spatial

L'évaluation d'une position non terminale au go est l'obstacle majeure de la programmation du jeu de go. Si l'on souhaite construire une fonction d'évaluation de positions arbitraires, donc non terminales, il est utile d'utiliser des concepts abstraits, comme Patrick Ricaud l'a montré dans son travail [127, 128]. Le résultat de mes recherches sur une fonction d'évaluation conceptuelle s'appliquant à des positions non terminales se résume à deux clefs : la reconnaissance des groupes et territoires avec la morphologie mathématique [50, 32] et à l'interaction ennemie [39, 41], présentée dans la sous partie 5.4.1. Une position de go contient beaucoup d'informations classifiables suivant des points de vues différents. Donc, il est difficile de résumer une position à une seule valeur. La sous-partie 5.4.2 présente les concepts spatiaux contenus dans une fonction d'évaluation au go [45]. La sous-partie 5.4.3 mentionne une publication présentant une étude de fonctions d'évaluation utilisant des principes métriques et des principes de dimension [47].

5.4.1 Morphologie mathématique et interaction ennemie : deux clefs

Premièrement, dans le travail de thèse, la morphologie mathématique, développée par Jean Serra [142], a été utilisée pour reconnaitre les "groupes", les "territoires" et l'influence d'une position [50, 32]. Ecrit de manière résumée, les groupes sont les fermetures morphologiques des pierres, les territoires sont les groupes moins les pierres et l'influence est la dilatation morphologique des pierres moins les groupes. Cela donne des résultats très bons quand on les compare aux groupes, territoires et influence reconnus par des joueurs humains. Indigo et GNU Go utilisent ce modèle. Deuxièmement, le principe de comparaison de deux groupes voisins ennemis a été défini et appelé "interaction". Au go,

cette notion est très importante. Pour la résumer, dans un combat entre deux groupes, le groupe qui a le plus de libertés ou le plus d'yeux gagne le combat. L'interaction est l'objet qui contient l'information sur le nombre d'yeux, de libertés communes ou libertés propres des deux groupes et contient la conclusion sur le gagnant du combat. [39, 41] sont deux publications qui mettent en valeur la notion d'interaction au go. Elles ont été mentionnées dans la sous-partie 5.1. Finalement, l'ensemble des ces concepts morphologiques et interactifs ont abouti à la fonction d'évaluation conceptuelle de Indigo. Ce travail s'intègre parfaitement dans le paradigme de la programmation de jeux suivant lequel la recherche arborescente appelle la fonction d'évaluation. L'implémentation de ces concepts a donné les versions de Indigo98 à Indigo2002. Bien que concurrencée par l'évaluation Monte Carlo développée récemment, cette fonction d'évaluation basée sur des concepts du domaine est toujours utilisée actuellement. Dans Indigo2003, couplée avec le générateur de coups, elle sert à effectuer la pré-sélection des coups donnés en entrée du module de Monte Carlo.

5.4.2 Le rôle des concepts spatiaux dans la fonction d'évaluation

L'avantage d'avoir une fonction d'évaluation complexe dans un domaine très visuel est d'être un terrain de recherches privilégié pour le raisonnement spatial [102, 84, 80]. La fonction d'évaluation au go est un des objectifs majeurs des recherches effectuées dans le projet Indigo, et un effort important y a été consacré. Ces recherches ont permis de découvrir, classifier, utiliser et tester des concepts spatiaux [45], métriques ou dimensionnels [47]. [45] part du constat que la difficulté de la programmation du go réside dans la complexité de sa fonction d'évaluation, celle-ci ayant une forte nature spatiale. Dès lors, il est intéressant de décrire les concepts spatiaux intervenant dans cette fonction pour en faire bénéficier la communauté du raisonnement spatial. Les concepts présentés dans [45] sont essentiellement l'encerclement, le territoire, le regroupement et le fractionnement. Les outils utilisés sont la morphologie mathématique et la distance de Hausdorff. La publication de [45] résulte de la participation à plusieurs groupes de travail sur le raisonnement spatial dans lesquels le rôle des concepts spatiaux dans la fonction d'évaluation ont été présentés : le groupe de travail du GDR I3 [3] sur les modèles spatio-temporels en 1998, le groupe de travail "Quando et Urbi" [161] organisé par Gérard Ligozat et Jean-Paul Sansonnet et enfin les journées nationales sur les modèles de raisonnement 1999 [4] organisée par le GDR I3.

5.4.3 Evaluations basées sur des concepts métriques et dimensionnels

[47] étudie les résultats d'outils basés sur la notion de distance ou sur la notion de dimension fractale. Ces outils sont utilisés par des programmes de go jouant sur des petits damiers et ils correspondent à des fonctions d'évaluation. La fonction d'évaluation basée sur la notion de distance reflète l'idée suivant laquelle un joueur veut minimiser la distance entre ses groupes de pierres et

maximiser la distance entre les groupes adverses. La fonction d'évaluation basée sur la notion de dimension fractale reflète plusieurs idées liées, pour chaque couleur, au nombre de groupes reconnus sur le damier, à leur taille et à l'espace total occupé. Ces outils sont évalués expérimentalement au travers de tournois organisés entre des populations de programmes paramétrés différemment. La méthode d'obtention des meilleurs programmes a été inspirée des algorithmes génétiques. A l'issue d'un tournoi, les meilleurs programmes sont gardés sans modification pour le tournoi suivant, d'autres sont gardés après avoir effectué une mutation. Les moins bons sont éliminés et remplacés par des nouveaux programmes paramétrés de manière à respecter l'équilibre entre exploration et exploitation. A la fin d'une série de tournois, la population converge vers un programme "optimal" dont les valeurs permettent d'évaluer le mérite relatif des outils basés sur la notion de distance par rapport à celui des outils basés sur la notion de dimension fractale.

5.5 Monte Carlo

Utiliser des statistiques pour avoir une fonction d'évaluation est une très bonne idée due à Abramson [7] et à Brügmann dans le cas du recuit simulé au go [59] et remise d'actualité par Bernard Helmstetter en 2002 lorsqu'il a commencé sa thèse sous la direction de Tristan Cazenave. Cette partie décrit le travail effectué dans le projet Indigo à propos de Monte Carlo. Elle traite d'abord de la méthode de Monte Carlo avec très peu de connaissances sur le go [56] (sous-partie 5.5.1), puis de l'association de Monte Carlo avec des connaissances sur le go [48] (sous-partie 5.5.2) et enfin de l'association de Monte Carlo avec de la recherche arborescente globale sur 9x9 (sous-partie 5.5.3). Monte Carlo est à mon avis la direction importante à suivre dans les années futures (sous-partie 5.5.4).

5.5.1 Monte Carlo avec très peu de connaissances

Les recherches décrites dans [56] partent de l'idée d'Abramson [7] et de celle de Brügmann [59]. Dans Abramson [7], l'idée est de calculer l'évaluation d'une position en lançant des parties aléatoires à partir de cette position et en les continuant jusqu'à la fin. Chaque partie aléatoire se termine sur une position facile à évaluer. L'évaluation de la position considérée est la moyenne des évaluations des positions finales des parties aléatoires. On peut aussi effectuer l'appel de cette fonction d'évaluation aux noeuds terminaux d'un arbre de profondeur N. Lorsque l'arbre est a profondeur 1, cela signifie que, pour chaque coup, on lance des parties aléatoires commençant par ce coup et finalement on joue le coup avec la meilleure évaluation moyenne. Dans Brügmann [59], l'idée est celle du recuit simulé. L'objet à optimiser est la séquence complète des coups de la partie. Au début du processus, la séquence est aléatoire, à la fin elle correspond dans une certaine mesure à une partie de go cohérente. Finalement, le programme joue le coup situé au début de la séquence. Bernard Helmstetter a surtout travaillé suivant cette approche en implémentant Oleg, un premier programme de Monte

Carlo. De mon côté, j'ai travaillé suivant l'idée d' Abramson en développant Olga, un autre programme de Monte Carlo. Dans Olga, l'idée d'Abramson a été enrichie de la notion du "progressive pruning", ce qui a été décrit dans [56]. Cette idée est classique dans les jeux contenant des informations cachées ou du hasard (cf. 6.3.2) tel que le poker [26], le backgammon [154, 155] ou le scrabble [144]. Elle consiste à maintenir à jour un intervalle de confiance autour de chaque moyenne calculée et à éliminer les coups dont l'intervalle de confiance est inférieur à celui du meilleur coup. Ainsi, les parties aléatoires ne sont lancées que sur les coups possibles statistiquement. Finalement, le plus surprenant était que Olga et Oleg, n'utilisant que très peu de connaissances, faisaient jeu égal sur 9x9 avec Indigo, basé sur beaucoup de connaissances. Dans [56], nous avons évalué expérimentalement des améliorations telles que "progressive pruning", "all-moves-as-first", la température constante ou décroissante dans le recuit simulé, et la profondeur 2. Les résultats de toutes ces améliorations sont bons sauf pour la profondeur 2. En effet, les résultats obtenus avec la profondeur 2 et un facteur de branchement de 80 ne sont pas concluants car il faut beaucoup de parties aléatoires pour réduire suffisamment l'écart-type des évaluations. Pour faire de la profondeur 2 ou plus sur les machines actuelles, il faut donc réduire le facteur de branchement avec des connaissances.

Par ailleurs, en collaboration avec Gilles Zémor de l'ENST, un projet de fin d'études sur Monte Carlo go a été suivi [68].

5.5.2 Monte Carlo avec beaucoup de connaissances

En observant que l'approche statistique et l'approche basée sur des connaissances donnent des résultats semblables sur 9x9, il était naturel de tester le niveau d'un programme combinant les deux approches. L'intégration d'une approche basée sur des connaissances avec cette approche statistique a été observée expérimentalement en faisant jouer trois versions d'Indigo : la version basée sur des connaissances, celle utilisant les statistiques et la meilleure, celle basée à la fois sur les connaissances et sur les statistiques. Les résultats quantitatifs des expériences sont décrits dans [48]. Les connaissances et les statistiques sont intégrées de deux manières. D'abord, le programme utilise la fonction d'évaluation conceptuelle et la génération de coups basées sur les connaissances du domaine. Ce pré-traitement produit 7 coups vérifiés tactiquement en entrée du module de Monte Carlo qui sélectionne alors un seul coup. De plus, pour déterminer l'urgence des coups au cours des parties aléatoires, le module de Monte Carlo n'utilise plus la probabilité uniforme mais une probabilité calculées à partir de patterns 3x3 et des chaînes en atari. En ce sens, les parties sont pseudo-aléatoires. Leur résultat final est plus significatif que celui des parties utilisant la probabilité uniforme. En conséquence, les moyennes calculées le sont aussi et le niveau du programme résultant augmente significativement. Ce travail a été décrit plus longuement dans [52], à paraître dans Information Sciences en 2004.

5.5.3 Monte Carlo avec de la recherche arborescente globale sélective

Afin de préparer Indigo spécifiquement pour les olympiades d'ordinateurs de go 9x9 en novembre 2003, un algorithme associant Monte Carlo avec de la recherche arborescente globale très sélective a été conçu. Un article décrit cette association [53]. Son contenu est décrit au paragraphe 5.3.2 de ce document.

5.5.4 L'intérêt de Monte Carlo pour la programmation du go

Il est utile de résumer pourquoi je pense que l'approche Monte Carlo est prometteuse. Deux obstacles principaux empêchent la programmation du go d'avancer. Le premier obstacle correspond à la difficulté à obtenir une fonction d'évaluation conceptuelle correcte et robuste, s'appliquant à des positions non terminales. Le second obstacle est la complexité combinatoire de la recherche arborescente globale. Si B est le facteur de branchement et L la longueur des parties, la complexité d'une recherche arborescente globale est en B^L. Il me semble que, dans une certaine mesure, Monte Carlo atténue ces deux obstacles. Premièrement, la solution Monte Carlo [7] offre une fonction d'évaluation statistique, très coûteuse, mais très robuste, contrairement à l'évaluation conceptuelle basée sur des connaissances. Les recherches statistiques s'appuient sur des évaluations de positions terminales (à la fin des parties aléatoires), elles sont donc fiables, même si le chemin pour y arriver est fait au hasard. Les évaluations terminales sont triviales et rapides à calculer. Deuxièmement, si N est le nombre de parties aléatoires lancées, la complexité de la recherche statistique globale est en $N \times B \times L$, donc nettement inférieure à celle de la recherche arborescente globale dans le cas du go. La puissance des machines actuelles rend possible l'approche Monte Carlo dés aujourd'hui. En conclusion, l'approche Monte Carlo réduit deux obstacles majeurs de la programmation du go en offrant une méthode statistique d'évaluation de positions non terminales et une recherche statistique globale donnant un niveau de confiance dans le coup sélectionné. Bien sûr elle ne résout pas le problème de la tactique locale, qui ne peut être résolu que par de la recherche arborescente.

6 Programmation des jeux de réflexion

Cette partie présente la programmation des jeux de réflexion : d'abord la programmation du jeu de go (sous-partie 6.1), ensuite la programmation des jeux de réflexion classique, c'est-à-dire celle des jeux d'échecs, checkers et othello dans laquelle on utilise de la recherche arborescente globale (sous-partie 6.2), ensuite celle non classique regroupant les jeux de Conway et les jeux à information incomplète (sous-partie 6.3), enfin celle du jeu d'amazones et du jeu d'hex (sous-partie 6.4). La sous-partie 6.1 sélectionne des articles sur la programmation du go puis résume *ma contribution à la programmation du go*. Les sous-parties 6.2 et 6.3 rassemblent une sélection d'articles de référence. Enfin, la sous-partie 6.4 relate *l'expérience d'encadrement acquise sur des projets d'étudiants* à propos de la programmation de jeux tels que amazones ou hex.

6.1 Programmation du jeu de go

Cette partie présente l'état de l'art de la programmation du jeu de go, puis la contribution du projet Indigo à ce domaine. L'état de l'art est présenté de manière très sélective. Seuls les articles utiles dans la programmation de Indigo ou les références indispensables sont citées par cette partie. La contribution est présentée de manière synthétique : liste des articles publiés et brièvement commentés, liste des résultats de Indigo à des compétitions internationales. Pour le détail de cette contribution, il est préférable de se reporter aux publications elles-mêmes.

6.1.1 Etat de l'art

Cet état de l'art de la programmation du go est présenté thème par thème : les articles généraux, la complexité combinatoire du problème, les meilleurs programmes, les petits damiers, la contribution du pionnier Albert Zobrist, les problèmes de vie et de mort et les réseaux de neurones. Cette partie fournit des références de qualité à de futurs programmeurs de go.

Articles généraux Les deux publications les plus récentes présentant un état de l'art en général sont [55] et [117]. [55] est un état de l'art orienté IA dans lequel les techniques de l'IA utiles pour la programmation du go sont répertoriées. [117] est classique et très bien structuré, écrit par un programmeur de référence, Martin Müller. [76], écrit par Ken Chen, est une classification des programmes de go actuels suivant le type du processus de décision utilisé : recherche globale, température, système expert ou autre. Dans le cadre de la programmation du go, [98] de Kierulf, Chen et Nievergelt est un article général sur l'ingénierie des connaissances et du logiciel. Enfin, il faut signaler la bibliographie en ligne [85], tenue à jour par Markus Enzenberger.

Complexité combinatoire du problème La complexité combinatoire du go a été étudiée il y a plus de vingt ans. [104] de Lichtenstein et Sipser montre que le go possède une difficulté polynomiale en espace en fonction de la taille du damier. [94] de Fraenkel et Lichtenstein montre que calculer une stratégie parfaite aux échecs, ou à d'autres jeux à information complète, joués sur damier de taille NxN est exponentiel en temps en fonction de N.

Les meilleurs programmes Goliath, meilleur programme au début des années 90, écrit par Mark Boon, est décrit dans [29]. L'algorithme de pattern matching de Goliath, utilisé dans Indigo, est décrit dans [28]. Go++ [125], le programme de Michael Reiss [126], meilleur programme actuel, n'est pas décrit scientifiquement, seulement dans un article du New Scientist paru en 2002 sur la programmation du jeu de go [58]. Go Intellect, écrit par Ken Chen, a été champion du monde une fois dans les années 90. Son processus de décision est décrit dans [74]. [79] contient la présentation d'heuristiques de go à propos de la vie et la

mort des groupes contenues dans Go Intellect et Goemate. Go Intellect est toujours honorablement classé dans les compétitions depuis plus de dix ans. David Fotland est l'auteur de Many Faces of Go [89], un des meilleurs programmes. David Fotland a décrit son premier programme G2 [90], écrit un algorithme de calcul de shichos [91], et publié un ensemble d'heuristiques sur la reconnaissance des yeux [92] utilisé dans Many Faces of Go. Many Faces a été champion du monde en 1998 et a gagné à Edmonton en 2002. KCC Igo, le programme nord-coréen, Goemate, le programme chinois du professeur Chen, Wulu, un programme dérivé de ce dernier, Haruka, le programme japonais de Ruichi Kawa, sont des programmes commerciaux qui participent avec succès aux compétitions internationales d'ordinateurs. Mais ils ne sont que très peu décrits, voire pas du tout. GNU Go [60] est le programme libre de la Free Software Fondation. Il a quasiment rattrapé ces meilleurs programmes. Il est développé par de nombreux programmeurs du monde entier. Daniel Bump dirige le projet. Inge Wallin et Günnar Farneback sont les principaux participants à la progression de GNU Go. Tous ces programmes ont un niveau difficile à évaluer sur l'échelle humaine : 8ème kyu[3] contre des adversaires n'utilisant pas les faiblesses du programme, mais seulement 15ème kyu contre des adversaires qui se le permettent.

Les petits damiers La recherche arborescente a permis la résolution du go 4x4 [141] par Sei et Kawashima en l'an 2000 et, associée à l'algorithme de Benson [18], celle du go 5x5 [164] par van der Werf en novembre 2002. Les deux études de Thorpe et Walden sur petits damiers [159, 160] restent intéressantes aujourd'hui.

Albert Zobrist Albert Zobrist a apporté deux célèbres contributions. D'abord à la programmation du jeu de go avec son modèle d'influence [175]. Ensuite à la programmation des jeux en général avec sa méthode de hachage des positions [176].

Les problèmes de vie et de mort Les problèmes de vie et de mort (tsumego) sont également l'objet de recherches particulières : l'algorithme de Benson [18] permet de calculer statiquement la vie et la mort de groupe. Il est utilisé dans certains programmes de go. GoTools, le programme spécifique de résolution de problèmes de vie et mort, écrit par Thomas Wolf, est décrit dans [171, 172, 173]. Il résout et engendre des problèmes de très haut niveau, environ 5ème dan. Mais ce n'est pas un programme jouant des parties complètes. Dans le but de jouer des parties complètes, la résolution de problèmes de vie et mort est utile pour les programmes décomposant le jeu global en sous-jeux. Si la décomposition est faite correctement, l'analyse de la vie est la mort des groupes par recherches arborescentes locales est pertinente.

[3] Un débutant est 30ème kyu, un joueur moyen est environ 10ème kyu, un joueur fort est 1er kyu ou 1er dan et les meilleurs joueurs professionnels sont 9ème dan.

Les réseaux de neurones Les réseaux de neurones [27] sont appliqués au go avec un succès mitigé actuellement. En 1994, Schraudolf, Dayan et Sejnowski utilisaient un réseau sur un damier sans autre caractéristique que la couleur des intersections [140]. Markus Enzenberger a appliqué un réseau dont les entrées correspondent aux chaînes de pierres et utilisent des caractéristiques simples, basées sur la distance de chaînes, les yeux, les libertés, les résultats de shichos, etc. Cela a donné le programme NeuroGo [86] qui a un niveau honorable [87]. [88] est la suite récente de NeuroGo, elle inclut un algorithme de calcul de connexion des groupes à l'intérieur même du réseau de neurones mais ses performances actuelles ne sont pas encore supérieure à celles de la version précédente de NeuroGo. Honte, de Fredrik Dahl, utilise, entre autre, des réseaux de neurones à plusieurs niveaux : à bas niveau pour proposer des coups, à niveau intermédiaire pour estimer la sécurité des groupes et au niveau stratégique pour évaluer les territoires [82].

6.1.2 Contribution du projet Indigo

La contribution du projet Indigo à la programmation du go se traduit par de nombreuses publications et à des participations aux compétitions d'ordinateurs avec des résultats toujours en progrès.

Publications depuis 1995 Dans les publications écrites depuis 1995, l'état de l'art orienté IA écrit avec Tristan Cazenave pour Artificial Intelligence Journal [55] est très souvent citée dans les introductions d'articles sur la programmation du go. Les publications sur l'association de Monte Carlo avec des connaissances [48, 52] et sur Monte Carlo seul [56] écrite avec Bernard Helmstetter reflètent le succès de cette approche. Par les réactions positives qu'elles suscitent au sein de la communauté des programmeurs de go, ces publications seront certainement utiles à la communauté de la programmation du go. Les publications sur la reconnaissance de territoires et d'influence [32, 50] ont déjà été utiles. Jérôme Dumonteil a participé au développement du programme libre GNU Go et a convaincu les membres de l'équipe de la pertinence de ce modèle. Les publications sur les fonctions d'évaluation [41, 45, 47] contribuent à surpasser l'obstacle majeur du domaine : trouver une bonne fonction d'évaluation. Des descriptions du fonctionnement interne de Indigo sont également disponibles [33, 35, 46, 51]. Les publications sur l'analyse rétrograde [44], l'incrémentalité [40], la représentation de l'incertitude [38], les sous-jeux [42] sont autant de points de vue sur un sujet complexe.

Participations aux compétitions depuis 1998 Le second aspect de la contribution à la programmation du go est "agonique" (du grec *agôn*, la confrontation, voir à ce sujet la classification des jeux faite par Roger Caillois [64]). A première vue une participation à un tournoi est une suite de confrontations, mais à seconde vue, c'est plutôt une suite de collaborations. En effet, à long terme, une partie de go jouée permet de tirer des enseignements positifs sur la façon de jouer du programme : son style général, ses points forts et ses points

faibles. Indigo a participé à plusieurs compétitions de programmes de go depuis 1998.

En ordre chronologique inverse, cela donne la liste suivante :
- 9x9 Computer Olympiads, Graz, Autriche, Novembre 2003 (4ème/10),
- 19x19 Computer Olympiads, Graz, Autriche, Novembre 2003 (5ème/11),
- Computer Go Festival, Guyang, Chine, Octobre 2002 (6ème/10),
- 21st Century Cup 2002, Edmonton, Canada, Juillet 2002 (10ème/14),
- 13x13 Stefan Mertin's tournament 2002, (15ème/19),
- 9x9 Stefan Mertin's tournament 2001, (7ème/11),
- Mind Sport Olympiad 2000, Londres, Aout 2000, (5ème/6),
- Ing Cup 1999 Shanghaï, Novembre 1999 (13ème/16),
- Computer Go Forum 1999, Tokyo, Juillet 1999 (24ème/28),
- Ing Cup 1998 Londres, Novembre 1998 (10ème/17).

Ces résultats se placent à un niveau international contre les meilleurs programmes mondiaux. Les deux derniers résultats (Indigo est classé dans la première moitié du classement) obtenus à Graz sur 9x9 [163] et sur 19x19 [78] sont très encourageants et ils devraient être suivis par d'autres bons résultats.

Pour répondre à l'objectif annoncé initialement d'obtenir un programme "aussi bon que possible", Indigo2003 est environ 12ème kyu, c'est-à-dire environ trois à quatre pierres en dessous des meilleurs programmes. A titre historique, le niveau de Indigo2002 est estimé à 15ème kyu, celui de Indigo99 à 20ème kyu et celui de Indigo95 à 25ème kyu.

Au cours des compétitions d'ordinateurs, généralement organisées en système suisse, Indigo s'est "confronté" de nombreuses fois des programmes de niveaux similaires. Il a rencontré SmartGo [97] de Anders Kierulf, GoLois [69] de Tristan Cazenave, Explorer [113] de Martin Müller, NeuroGo [86] de Markus Enzenberger, Aya [174] de Hiroshi Yamashita, et, par là, un respect mutuel s'est dégagé entre les auteurs de ces programmes. Indigo a également joué contre des programmes plus forts : Go4++ [125] de Mick Reiss, Many Faces [89] de David Fotland, Goemate du professeur Chen, GNU Go [60] de la FSF, GoAhead [170] de Peter Woitke, Go Intellect de Ken Chen.

6.2 Programmation classique des jeux de réflexion

Cette partie présente la programmation classique des jeux de réflexion en général puis celle des jeux classiques, c'est-à-dire utilisant de la recherche arborescente globale.

6.2.1 Jeux de réflexion en général

Les jeux ont été étudiés depuis plus soixante ans : Von Neumann et Morgenstern avait déjà étudié les arbres min-max [166]. Récemment, Schaeffer a fait une revue des avancées de la programmation des jeux de réflexion [136]. La thèse de Allis est également un document très intéressant pour connaître l'état de résolution des jeux [10]. Dans le numéro spécial de AI Journal, Jonathan

Schaeffer et Jaap van den Herik dressent un état de l'art du niveau des programmes de jeux de réflexion [139]. Jaap van den Herik, Jos Uiterwijk et Jack van Rijswijck font des pronostics sur les progrès futurs dans ces jeux [162].

6.2.2 Echecs, othello, checkers

La programmation des échecs, d'othello et des checkers est presque synonyme de recherche arborescente déjà longuement décrite au 5.3. Un des premiers articles sur la programmation des échecs est celui de Claude Shannon [143]. Un des plus récents est [65] dans lequel, avec 5 ans de recul, Murray Campbell décrit le développement de Deep Blue et sa victoire sur Kasparov. Mais en 2004, où en sont les programmes d'échecs ? D'une part, Kasparov a fait match nul 2-2 contre Fritz, un des meilleurs programmes d'échecs actuel en novembre 2003. D'autre part, Fritz a terminé 1er ex aequo avec Shredder aux derniers championnats du monde à Graz également en novembre 2003. Ce qui montre que, même si Deep Blue s'est retiré, un nombre croissant de programmes d'échecs ont le niveau des meilleurs joueurs humains et remplace Deep Blue avec succès.

La programmation d'othello a été marquée par Logistello de Michael Buro [61, 62]. En 1997, Logistello était déjà largement supérieur au meilleur joueur humain. La programmation des checkers a commencé avec les travaux célèbres de Samuel [131, 132]. Jonathan Schaeffer est l'auteur de Chinook. L'histoire de son développement et de sa confrontation contre Marion Tinsley est décrite en détails dans [135]. Les meilleurs programmes de checkers sont donc maintenant meilleurs que les meilleurs humains. Des considérations à propos de la résolution future des checkers figurent dans [137]. Cependant, aujourd'hui, Jonathan Schaeffer estime que la résolution des checkers n'est pas imminente.

6.3 Programmation des jeux vue d'ailleurs

Cette partie présente la programmation des jeux de réflexion vue d'un point de vue différent de celui des jeux à information complète utilisant la recherche arborescente : la théorie des jeux sommables de Conway et la programmation des jeux à information incomplète.

6.3.1 Jeux combinatoires de Conway

Les deux références de base sont "On Numbers and Games" [81] de John Conway, livre assez formel, et "Winning Ways" de Berlekamp, Conway et Guy [22], livre avec de nombreux exemples et notamment la présentation des stratégies de jeu basées sur la thermographie. Fraenkel a publié une définition constructive d'un jeu dans "What is a game ?" [93]. Il part du jeu 0 créé au jour 0, des trois jeux *, 1 et -1 créé le jour 1 à partir du jeu 0 et le droit de jouer un coup. Au jour N, seuls les jeux permettant d'arriver à un jeu du jour N-1 en un coup sont considérés. Une introduction à "mathematical go" est donnée dans l'article de Berlekamp de 1991 [19] et un livre sur le même sujet [23] a été publié en 1994 après le succès de mathematical go sur des positions adhoc où

des professionnels de go trouvaient des résultats inférieurs. L'étude sur les fins de parties au go [118] de Martin Müller et Ralph Gasser donne un autre aperçu de ce qu'il est possible de faire avec les jeux de Conway au go. Landman décrit les yeux au go suivant le formalisme des jeux combinatoires [103]. L'article de Bill Spight relatant une partie de go dans laquelle les deux joueurs, Jiang et Rui, ont le choix entre jouer un coup sur le damier ou prendre un ticket valant un certain nombre de points [147] est très intéressant. La suite des valeurs entières des tickets est décroissante de N points à 1 point. Les deux joueurs jouent la partie de go et, de temps en temps, au lieu de jouer un coup, ils choisissent de prendre un ticket de points. Par ailleurs, l'article de Berlekamp décrivant une partie dans laquelle les deux joueurs ont le choix de jouer un coup dans un des 4 jeux suivants : go, domineering, dames, échecs [21] est aussi très intéressante. Les deux joueurs utilisent la théorie des jeux combinatoires pour effectuer leurs choix. Le résultat est une "partie" étonnante dans laquelle des (sous-)jeux très différents sont traités avec le même outil.

En résumé, les jeux combinatoires offrent un cadre formel utile pour décrire des aspects ludiques du jeu de go et d'autres jeux. C'est un outil mathématique utile dans le contexte de la décomposition du jeu global en sous-jeux.

6.3.2 Jeux contenant du hasard ou à information incomplète

La programmation des jeux contenant explicitement du hasard (backgammon) ou des informations cachées (poker, bridge, scrabble) inclut naturellement des simulations. La fonction "random" de l'ordinateur est utile pour simuler les lancements de dé, le choix des tuiles ou les cartes cachées. A la lecture de [26, 154, 144], les simulations donnent de très bons résultats à ces jeux, ce qui est une source d'inspiration pour tester aussi les simulations au go, même si celui-ci ne contient pas explicitement de hasard ni d'information cachée dans la définition de ses règles. Néanmoins, lorsque deux joueurs d'un jeu suffisamment complexe s'affrontent, ils utilisent chacun leur propre modèle de la position. Donc chaque joueur possède des informations que l'autre ne connaît pas. Donc un jeu suffisamment complexe est en fait un jeu implicitement à information incomplète ou cachée. De ce point de vue, il est adapté d'utiliser des simulations dans ces jeux, ce qui a été fait au go avec Bernard Helmstetter et que Abramson avait déjà fait sur othello en 1990 [7].

6.4 Encadrement de projets de programmation de jeux de réflexion

Cette partie présente l'expérience d'encadrement de projets ou stages sur la programmation de jeux de réflexion : le jeu d'amazones et le jeu d'hex.

6.4.1 Amazones

Amazones [152] est un jeu de nature intermédiaire entre celle des échecs et celle du go. Des amazones se déplacent sur un damier 10x10, de la même

manière que les reines aux échecs, et elles lancent des flèches sur les cases du damier, les rendant inaccessibles. Les joueurs jouent à tour de rôle en effectuant un déplacement d'une amazone et un lancer de flèche. Lorsqu'un joueur ne peut pas jouer, il a perdu. Ce jeu ressemble au go car des zones entourées par des flèches apparaissent et elles deviennent des "territoires" lorsqu'elles sont occupées par des amazones d'une seule couleur. Müller et Tegos décrivent les objets et méthodes pertinents à amazones [119]. Snatzke a étudié le résultat de la recherche exhaustive sur des petits damiers [146]. La fonction d'évaluation de Amazong [105], le meilleur programme d'amazones, de Jens Lieberum, est décrite dans [106].

Le jeu d'amazones ressemble un peu au jeu de l'ange et du diable défini par Berlekamp [20]. Sur un damier de taille infinie se trouve un ange pouvant se déplacer comme une reine à amazones d'une distance inférieure à sa "puissance" (si la puissance égale 1, c'est le déplacement du roi aux échecs). A tour de rôle, l'ange se déplace et le diable lance une flèche sur le damier, la rendant inaccessible. Le but de l'ange est de rester libre tout le temps et le but du diable est d'encercler l'ange. Il a été montré que l'ange de puissance 1 est capturable mais rien n'a été démontré pour un ange de taille supérieure ou égale à 2. Une variante de ce jeu s'appelle DukeGo. Elle se joue sur damier de taille finie et [110] est une présentation des résultats actuels.

Un coup d'amazones se décompose en deux sous-coups : d'abord déplacer l'amazone puis lancer la flèche. Il en résulte deux façons d'engendrer les parties aléatoires, et aussi deux façons de faire le Monte carlo à profondeur un. Concernant la génération aléatoire de coups, il y a une méthode "lente" correspondant à la probabilité uniforme et une méthode "rapide" ne lui correspondant pas. Dans la méthode lente, on engendre classiquement tous les coups à profondeur 1, ce qui oblige le générateur de coups à déplacer des amazones pour déterminer où lancer les flèches, puis *à replacer les amazones*, ce qui pourrait être évité. Chaque coup est stocké dans un tableau et on choisit ensuite le coup aléatoirement. Dans la méthode rapide, on choisit d'abord au hasard le déplacement de l'amazone, puis ce déplacement étant connu, on choisit la destination de la flèche au hasard. On évite donc le retour arrière des amazones.

De plus, la méthode de Monte Carlo peut être aussi décomposée en deux phases : la première phase consistant à faire des moyennes sur les parties aléatoires sachant que telle amazone va à telle destination afin de choisir l'amazone et sa meilleure destination. La seconde phase, connaissant l'amazone et sa destination, consiste à faire des moyennes sur les parties aléatoires sachant que la flèche est lancée à telle destination afin de choisir la meilleur destination de la flèche. Ainsi on diminue largement le nombre de partie aléatoires à simuler. Malheureusement, cette méthode en deux phases perd en niveau de jeu.

En 2003 et 2004, j'ai encadré des étudiants pour appliquer Monte Carlo au jeu d'amazones. A l'été 2003, Arnaud Caillieretz a développé un premier programme. A l'automne 2003, j'étais responsable de l'UE projet "Programming and Statistics" de la MST ISASH. Durant cette UE, les étudiants ont programmé la méthode de Monte Carlo suivant différentes variantes. De même, dans le cours de programmation orienté objet du magistère MIAIF, le projet demandé

consistait en l'écriture de cette méthode pour amazones.

Les étudiants de MST ISASH ont effectué des expériences montrant l'importance du nombre d'itérations, l'importance de jouer en premier, l'importance relative de différentes fonctions d'évaluation (qui a gagné, combien de coups restants en profondeur pour le gagnant, combien de coups restants en largeur), l'importance de l'élagage progressif des coups (progressive pruning), les avantages et inconvénients d'utiliser la méthode de génération aléatoire "rapide" ou "lente", et le pour et le contre de la méthode de Monte Carlo à profondeur un décomposée en deux phases. Les rapports des projets les plus intéressants sont disponibles [67, 25, 108, 8].

Les étudiants de magistère MIAIF, meilleurs en programmation, ont créés des joueurs Monte Carlo utilisant des parties aléatoires plus optimisés en temps de réponse, mais ils n'ont pas fait les expériences sur l'importance des différents paramètres de la méthode, ce qui n'était pas leur objectif.

6.4.2 Hex

Hex [13] est un jeu dans lequel les joueurs essaient de connecter leurs bords. Hex se joue sur un damier hexagonal. Une description du programme Queenbee de Jack van Rijswijck [165] avec une fonction d'évaluation intéressante basée sur la notion de "2-distance". Hexy est le meilleur programme actuel, basé sur des connexions virtuelles [14]. J'ai encadré un stagiaire, Vincent Airault, pendant 4 mois sur le jeu d'hex. Son approche visait à appliquer les algorithmes génétiques [9]. Cela a donné un programme d'un niveau "intéressant" mais plus faible que Hexy ou Queenbee.

7 Autres activités liées à la recherche

Cette partie regroupe les activités liées à la recherche académique auxquelles j'ai participé : les séminaires et groupes de travail, les activités scientifiques sous-jacentes aux conférences et journaux et l'encadrement de stagiaires.

7.1 Présentations dans des séminaires et groupes de travail

Hormis les conférences avec actes publiés, je présente régulièrement mes activités de recherche dans des séminaires ou des réunions scientifiques et j'ai animé une session d'une conférence proche de mes thèmes de recherche :

- GDR I3 [3] Modèles spatio-temporels : "Programmation du go et raisonnement spatial", 3 décembre 1998,
- Groupe "Quando et Urbi" [161], animé par Gérard Ligozat et Jean-Paul Sansonnet, idem, 9 mars 1999,
- Journées Nationales sur les Modèles de raisonnement, JNMR99 [4], idem, mars 1999,

- Conseil scientifique de l'UFR mathématiques et informatique de Paris 5, "Le projet Indigo", mars 2000,
- Animateur de la session "Art et Jeu" de "CJC-03" (Colloque Jeunes Chercheurs en Sciences Cognitives), juin 2003,
- Journée CRIP5, "Le projet Indigo", 26 janvier 2004,
- Groupe "IA et jeu" animé par Vincent Corruble au LIP6, "Le projet Indigo", 27 février 2004.

7.2 Contributions annexes à la recherche académique

Dans le cadre de la recherche académique, je participe à des activités sous-jacentes d'organisation ou de relecture. Voici la liste de mes contributions :

- Membre du comité de programme de la conférence "Computational Intelligence in Games", CIG-05 [107],
- Rapporteur d'articles des conférences internationales "Computers and Games", "Advance in Computer Game 10", "ECAI-2000" et de l'International Computer Game Association Journal,
- Examinateur dans le jury de la thèse de Régis Monneret [112], juillet 2000.

Devant la croissance du nombre de publications à propos de la programmation des jeux de réflexion, le calendrier des conférences dans ce domaine s'est étoffé régulièrement ces dernières années. En 1998, la conférence biennale "Computers and Games" a été créée et se tient désormais les années paires. La série de conférences quadriennale "Advance in Computer Games" obtient toujours un grand succès. Elle se tient tous les quatre ans, la dernière ayant eu lieu en 2003. Tous les deux ans au moins, un "Game Programming Workshop" est organisé par le Japon. Par ailleurs, le département informatique de l'université de Maastricht organise un workshop associé aux olympiades d'ordinateurs quand celles-ci ne sont pas déjà associée à une grande conférence du domaine (CG ou ACG). Les conférences CG et ACG ont toujours été des francs succès mais elles ne couvrent pas toutes les années. Devant la demande des chercheurs, une nouvelle conférence est créée en 2005 : "Computational Intelligence in Games" (CIG-05). Je fais partie du comité de programme de cette conférence.

7.3 Encadrement de stagiaires

Cette partie rassemble les stages encadrés depuis 1992. Pendant ma thèse, en collaboration avec Jacques Pitrat, j'ai encadré deux stagiaires de DEA à l'université Paris 6 : Pierre Recoque et Tristan Cazenave. A l'université Paris 5, j'ai encadré Vincent Airault, étudiant de magistère MIAIF. En 2003, en collaboration avec Gilles Zémor, j'ai supervisé les projets de fin d'études de Jean-François Goudou et Vincent Castillo, étudiants de l'Ecole Nationale Supérieure des Télécommunications (ENST). J'ai récemment dirigé Arnaud Caillieretz et Christophe Truong, étudiants de l'Institut des Sciences et Techniques (IST), rattaché à l'université Paris 6. Je vais diriger Guillaume Chaslot, étudiant à l'Ecole Centrale de Lille, à l'été 2004.

Pierre Recoque Pattern-matching au jeu de Go (DEA IARFA, Université Paris 6, 1992). Ce stage co-encadré avec Jacques Pitrat a permis à Pierre Recoque d'écrire un algorithme de pattern-matching utile pour le go.

Tristan Cazenave Création automatique de règles à patterns (DEA IARFA, Université Paris 6, 1993). Ce stage co-encadré avec Jacques Pitrat a permis a Tristan Cazenave de développer un premier outil de génération automatique utile pour la thèse qu'il a effectué ensuite.

Vincent Airault [9] De l'application de méthodes classiques au jeu d'Hex vers une utilisation des algorithmes génétiques (MIAIF2, Université Paris 5, 2001). Ce stage a permis a Vincent Airault de se familiariser avec les techniques de recherche arborescente et ensuite de voir comment les algorithmes génétiques peuvent s'appliquer au go.

Jean-François Goudou, Vincent Castillo Monte Carlo Go (ENST, Printemps 2003) [68]. Ce projet de fin détudes co-dirigé avec Gilles Zémor consistait à reproduire l'expérience Monte Carlo go. Ce qui a été fait avec le succès attendu.

Arnaud Caillieretz Monte Carlo Amazones (Institut des Sciences et Techniques (IST), Université Paris 6, Eté 2003). Ce stage a constitué une première tentative d'application de Monte Carlo au jeu d'amazones. Ce travail a ensuite servi de base de départ pour l'encadrement des projets "programmation et statistiques" de MST ISASH2 à l'automne 2003.

Christophe Truong Monte Carlo Echecs (Institut des Sciences et Techniques (IST), Université Paris 6, Eté 2003). Ce stage a constitué une première tentative d'application de Monte Carlo au jeu d'échecs. Même si ce travail était intéressant, la technique s'est révélée peu concluante, ce qui était le résultat attendu.

8 Perspectives

Les perspectives des recherches présentées sont nombreuses. Elles concernent essentiellement la prolongation du travail utilisant Monte Carlo mais aussi la re-génération automatique des bases de formes utilisées par Indigo, l'application de techniques d'apprentissage non supervisé telle que la méthode des différences temporelles, le parallélisme et une ouverture vers d'autres jeux de réflexion tels que le jeu d'amazones.

8.1 Construction automatique de patterns suivant une approche Bayésienne

La première perspective est de remplacer les bases de patterns acquises manuellement par des bases acquises automatiquement. Ce travail a déjà com-

mencé. Une approche consiste à faire de l'apprentissage bayésien de patterns du type K plus proches voisins [27] en utilisant des parties de professionnels disponibles du Internet.

Le pré-processeur actuel donnant les coups à évaluer à Monte Carlo utilise trois bases de formes distinctes répondant à trois besoins différents. La première base est composée de formes utilisées par l'évaluation de la force des groupes dans la fonction d'évaluation. La seconde base est composée de formes "de coin", ou josekis[4] simples, et la troisième est composée de formes "de bord". Les formes de ces trois bases ont le point commun d'être assez générales et peu nombreuses. Elles ne contiennent que peu de pierres. L'idée de remplacer ces trois bases par une base unique reposant sur l'approche des K plus proches voisins est tentante. L'implémentation de cette idée a été commencée pour la base de formes utilisée par la fonction d'évaluation et cette idée sera également poursuivie pour les deux autres bases. L'apprentissage envisagé ici est totalement supervisé par les exemples des positions contenues dans les parties de professionnels disponibles sur Internet. Le calcul des urgences des formes sera fait avec la formule de Bayes en calculant des proportions de coups joués ou pas, et de patterns reconnus ou pas au cours de ces parties. Il y a deux probabilités pertinentes à calculer : en cours de jeu, la probabilité de jouer un coup conseillé par un pattern sachant que celui-ci a été reconnu, et en cours d'apprentissage, la probabilité de reconnaître un pattern conseillant un coup sachant qu'il a été joué par un professionnel.

8.2 Suite de Monte Carlo go

Etant donnés les résultats encourageants obtenus en 2003, la perspective naturelle est de continuer l'approche Monte Carlo. Aujourd'hui, la base de formes 3x3 utilisée par le moteur de parties aléatoires a été construite de deux manières. La première, peu coûteuse, a consisté à reprendre les formes de taille inférieure à 3x3 dans la base de formes utilisée par la fonction d'évaluation de Indigo. Cette première approche est manuelle au sens où les formes utilisées ont été rajoutées à la main au fil du temps. La seconde approche a consisté à engendrer cette base suivant l'approche bayésienne décrite ci-dessus. Malheureusement, le nombre de parties de professionnels utilisées actuellement (360) n'est pas suffisant pour calculer des probabilités significatives, une bonne forme, "fréquente" et "urgente", n'apparaît que quelques fois en cours de parties. Cette expérience est à refaire avec "beaucoup" de parties. Une troisième direction est de faire de l'apprentissage par renforcement [150], ou plus précisément utiliser l'idée du "Q-learning" de Watkins [167, 168], pour améliorer l'urgence des patterns 3x3 utilisés pendant les parties pseudo-aléatoires. En plus de son autonomie et de son indépendance vis-à-vis de l'extraction des parties de professionnels, l'avantage de cette approche est son adéquation au problème : trouver des urgences de patterns, autrement dit des valeurs d'actions au sens de [150]. Il ne s'agit pas de trouver le pattern correct à sélectionner dans une position, ce que fait l'apprentissage supervisé.

[4] séquence de coups jouée au début de la partie dans un coin du damier donnant un résultat équilibré.

Puisque les épisodes Monte Carlo donnent de bons résultats lorsqu'ils correspondent à des parties complètes, il peut être intéressant de faire des épisodes plus courts et plus conceptuels. Tesauro et Galperin [155] ont réalisé cette expérience au backgammon. Par exemple, faire des épisodes longs de 10 coups en appelant, ou bien la fonction d'évaluation conceptuelle à la fin de la séquence (première approche), ou bien la fonction d'évaluation Monte Carlo basique (deuxième approche). La première approche sera intéressante si le fait de faire des moyennes sur des épisodes permet de réduire l'erreur contenue dans la fonction d'évaluation conceptuelle et si le coût des épisodes conceptuels est inférieur à celui des épisodes basiques actuels. Dans la seconde approche, un coup sera engendré en faisant de la recherche arborescente à profondeur 1, 2 ou 3, en lançant des épisodes conceptuels de quelques coups, suivis par des épisodes de bas niveau correspondant à l'approche actuelle. Cette seconde approche peut donner des résultats, les épisodes conceptuels permettant un passage progressif de la recherche arborescente min-max à la recherche statistique en moyenne. Dans les deux approches, le problème à résoudre est une fois de plus celui de la rapidité de la génération de coups conceptuels. Dans la première approche, avoir une fonction d'évaluation conceptuelle rapide est nécessaire.

8.3 Parallélisme

A partir du moment où le point critique du programme est la vitesse du moteur de parties aléatoires, étudier les approches parallèles me semble opportun. Tesauro et Galperin ont effectué leurs expériences sur Monte Carlo au backgammon [155] en utilisant une approche parallèle.

8.4 Appliquer les différences temporelles au go

L'algorithme des différences temporelles [149] de Richard Sutton a été appliqué avec succès au backgammon par Tesauro [154, 153]. Au go, Dayan et Sejnowski l'ont appliqué tel quel sans utiliser de connaissances particulières [140]. Ensuite, Markus Enzenberger l'a également utilisé, mais en ajoutant des connaissances sur le go, ce qui a donné NeuroGo [87], un programme d'un niveau intéressant, étant donnée son approche. L'algorithme des différences temporelles appliqué au domaine des jeux consiste à calculer la fonction d'évaluation du jeu avec l'aide d'un approximateur de fonction, par exemple un réseau de neurones. Le réseau de neurones apprend la fonction d'évaluation sur des exemples de positions de parties jouées à l'envers. La valeur d'une position étant recalculée à partir de son ancienne valeur plus un facteur de correction proportionnel à l'erreur entre la sortie du réseau de neurones et une valeur donnée par l'algorithme des différences temporelles. Cette approche est très tentante. Cependant, vouloir appliquer les différences temporelles au go signifie, entre autres choses, vouloir obtenir une bonne fonction d'évaluation automatiquement. Or Monte Carlo répond déjà à la question, même si son coût d'exécution est très cher. Ensuite, les différences temporelles doivent être appliquées pour faire mieux que la fonction d'évaluation actuelle construite à la main. En conclusion, la méthode

des différences temporelles offre une perspective intéressante par son approche automatique, à comparer avec l'existant déjà construit manuellement.

8.5 Développer Monte Carlo amazones

En 2003, un étudiant de l'IST à Paris 6 a effectué un stage sur Monte Carlo amazones. Dans l'unité d'enseignement "Informatique et Statistiques" de la MST ISASH et en Magistère MIAIF, des projets sur Monte Carlo amazones ont été dirigés. Je souhaite poursuivre cette activité d'encadrement. Pour le go, l'approche a été d'abord basée sur les connaissances pendant longtemps, puis sur Monte Carlo. En revanche pour amazones, la stratégie du jeu est peu développée car le jeu est récent. La chronologie de l'approche sera donc opposée de celle du go : d'abord tester l'approche Monte Carlo, puis l'enrichir avec des connaissances sur amazones. Une collaboration avec Jens Lieberum, auteur du meilleur programme d'amazones actuel, Amazong [106], pour voir l'apport des simulations Monte Carlo à ce jeu est envisagée.

9 Conclusion

Dans le but d'obtenir une habilitation à diriger des recherches, ce document a présenté les travaux de recherche effectués sur la programmation du jeu de go depuis 1997 dans groupe SBC (Systèmes à Bases de Connaissances) de l'équipe IAA (Intelligence Artificielle et Applications), dirigée par Dominique Pastre à l'université Paris 5. Dans cette conclusion, le but est de montrer dans quelle mesure les objectifs caractéristiques d'une habilitation à diriger des recherches ont été atteints : suivre une démarche originale dans un domaine scientifique, maîtriser une stratégie autonome de recherche scientifique et démontrer une capacité d'encadrement de jeunes chercheurs.

Après avoir rappelé pourquoi la programmation du go est intéressante pour l'IA, l'objectif des recherches est d'obtenir un programme de go aussi bon que possible, en validant les progrès du programme par la participation à des compétitions d'ordinateurs, et en publiant les résultats des recherches. La démarche est basée sur un principe expérimental et un principe didactique. Le principe expérimental consiste à ne garder que les idées validées par les expériences et à avoir à tout moment un prototype regroupant toutes ces idées. Le principe didactique consiste à varier les approches régulièrement. Cela permet de ne pas rester bloqué longtemps sur un problème, et donc de toujours avancer. Cela permet aussi d'acquérir de nouvelles connaissances et de se former à des techniques nouvelles pour trouver ensuite. Cela permet enfin d'obtenir des certitudes sur la pertinence ou l'inadéquation des idées à utiliser pour résoudre un problème. Ceci constitue ma démarche suivie dans le domaine de la programmation des jeux de réflexion et l'intelligence artificielle. Elle a eu le mérite d'aboutir à des résultats expérimentaux concrets.

Puis, un historique du développement de Indigo a été montré période par période. La période 1997-1999 a permis de ré-écrire totalement le programme

Indigo en ayant une fonction d'évaluation globale avec de la recherche arborescente à profondeur 1. La période 2000-2002 a permis de me familiariser avec les spécificités de la recherche arborescente (améliorations de alfa-béta) et d'améliorer Indigo itérativement, les faiblesses de l'année précédente étant corrigées l'année en cours. En 2003, l'approche Monte Carlo a beaucoup simplifié Indigo. Les recherches arborescentes de niveau global présentes dans la version 2002 disparaissent. Cet historique montre que la démarche est également tenace et persévérante. En suivant une approche de manière régulière, il est possible d'arriver à de très bons résultats. C'est peut-être ce qui fait l'originalité de la démarche.

Ensuite, des éclairages différents sur mes publications ont été donnés, suivant les thèmes de l'IA, et suivant la programmation des jeux. [55], écrit avec Tristan Cazenave pour AI Journal, est très souvent cité dans les introductions des articles sur la programmation du go. L'article pour la revue IJPRAI [50] décrit la reconnaissance de territoires et d'influence en utilisant la morphologie mathématique. L'algorithme de reconnaissance de territoires s'est répandu au sein de la communauté des programmeurs de go sous le nom de "l'algorithme 5-21 de Bouzy" et il est utilisé par GNU Go. Les travaux sur les fonctions d'évaluation décrivent des connaissances dépendantes du domaine considéré, le go. Le domaine du raisonnement spatial a permis de mettre en valeur ces travaux, notamment au travers des groupes de travail, ce qui a abouti finalement à un article dans la revue d'intelligence artificielle [45]. La communauté de l'IAD a également permis de valoriser le concept d'interaction [41], pertinent au go et en IAD. La communauté de la programmation des jeux de réflexion, qui s'est significativement élargie au niveau académique depuis la victoire de Deep Blue sur Kasparov, a permis de publier un article sur les concepts de distance et dimension au go [47]. Le processus de décision de Indigo 2002 est décrit dans [51]. Monte Carlo est venu simplifier ce processus en 2003. L'avenir confirmera ou pas la pertinence de l'approche Monte Carlo entamée avec Bernard Helmstetter [56] et complétée avec des connaissances [48, 52] ou avec de la recherche arborescente globale [53]. Ce qui est certain, c'est que cette approche sera de plus en plus intéressante au fur et à mesure que la puissance des machines augmentera. Donc, le résultat de mes recherches en termes de publications et de résultats dans des compétitions d'ordinateurs démontre la maîtrise de la stratégie autonome de recherche scientifique que j'ai atteinte.

Dans la partie 7, j'ai listé l'ensemble des activités indirectes de recherche scientifique : la participation à un comité de programme d'une conférence internationale, l'animation d'une table ronde dans un colloque pour jeunes chercheurs, la relecture d'articles pour des conférences internationales et l'encadrement d'étudiants lors de nombreux stages ou projets de fins d'études directement liés à mon domaine scientifique (sous-parties 6.4 et 7.3). Par là, j'ai donc montré ma capacité d'encadrement de jeunes chercheurs.

A l'avenir, l'approche statistique sera encore utilisée pour améliorer Indigo : approche bayésienne [27] ou apprentissage par renforcement [150]. Une approche parallèle, la méthode des différences temporelles et le jeu d'amazones sont d'autres perspectives intéressantes à développer, et ainsi faire profiter les

communautés de l'IA et de la programmation des jeux de réflexion d'autres publications.

Finalement, par la capacité d'encadrement de jeunes chercheurs démontrée lors de stages, par la démarche originale, expérimentale et didactique, suivie dans le domaine de la programmation des jeux de réflexion, par la maîtrise d'une stratégie autonome de recherche scientifique, témoignée par de nombreuses publications et des résultats honorables dans les compétitions internationales d'ordinateurs, j'espère avoir donné les éléments permettant d'obtenir une habilitation à diriger des recherches en informatique.

10 Remerciements

Je remercie Dominique Pastre, professeur d'informatique à l'université Paris 5, de m'avoir accueilli dans l'équipe SBC en 1997, et d'avoir suivi mes recherches avec beaucoup d'attention. Même si mes recherches diffèrent sensiblement de son sujet de prédilection, l'automatisation du raisonnement mathématique, au sein de l'IA elles partagent la même attitude : la résolution d'un problème complexe.

Je remercie Patrick Greussay, professeur d'informatique à l'université Paris 8, pour son intérêt, entre autres choses, pour la programmation du go et son accueil dans le laboratoire d'IA de Paris 8 au printemps 1997, avant que ne je sois nommé à Paris 5.

Je remercie Jacques Pitrat, directeur de recherche émérite en informatique au CNRS, responsable de l'équipe Métaconnaissances du LIP6, pour la direction de ma thèse entre 1991 et 1995 et son intérêt pour la programmation des jeux de réflexion, support privilégié des recherches en IA.

Je remercie Bernard Victorri, directeur de recherche en informatique au CNRS au laboratoire LATTICE associé à l'Ecole Normale Supérieure, d'avoir encouragé, facilité et validé mon travail de thèse en 1995, et plus généralement tous les travaux sur la programmation du go.

Je remercie Gérard Ligozat, professeur d'informatique à l'université Paris 11 au LIMSI, et Mario Borillo, directeur de recherche CNRS en informatique à l'IRIT, pour leur ouverture d'esprit lorsque j'ai travaillé sur le raisonnement spatial au go.

Je remercie Jean Michel, directeur de recherche en mathématiques au CNRS, un des meilleurs joueurs de go français, passionné par les progrès des programmes de go. Il offre un regard critique rigoureux indispensable sur les programmes de go.

Je remercie Jean Serra, directeur de recherche à l'école des Mines de Paris, directeur du centre de morphologie mathématique, et fondateur (avec Georges Matheron) de la morphologie mathématique.

Je remercie Ken Chen, professeur d'informatique à l'université de Caroline du Nord, auteur de Go Intellect, référence de la programmation du go, organisateur de la session sur la programmation des jeux et la recherche heuristique au sein de la conférence JCIS, pour sa compétence et sa sympathie envers mon travail.

Je remercie Martin Müller, professeur d'informatique à l'université d'Alberta, auteur de Explorer, référence de la programmation du jeu de go, pour la clarté de ses idées, ses travaux sur la programmation du go et l'application de la théorie des jeux combinatoires.

Je remercie Michael Buro, professeur d'informatique à l'université d'Alberta, auteur de Logistello, le meilleur programme d'othello, pour ses idées, ses discussions et son regard rigoureux sur la programmation des jeux en général.

Je remercie Jaap van den Herik, professeur d'informatique à l'université de Maastricht, directeur de l'IKAT (Institute for Knowledge and Agent Technology), Jonathan Schaeffer, professeur d'informatique à l'université d'Alberta, directeur du "GAMES group", et Hiroyuki Iida, professeur d'informatique à l'université de Shizuoka, pour le cadre académique international fort qu'ils développent autour de l'ICGA.

Je remercie Tristan Cazenave, maître de conférences en informatique à l'université Paris 8, habilité à diriger des recherches, pour sa compétence, son programme GoLois, et son approche originale orientée vers la tactique.

Je remercie particulièrement Bernard Helmstetter, actuellement doctorant en informatique sous la direction de Tristan Cazenave, pour son regard neuf sur la programmation du go, son programme Oleg, et le regain qu'il a déclenché chez moi sur l'approche Monte Carlo. Par ailleurs, je le félicite pour son titre de champion de France de go acquis en 2003.

Je remercie Anders Kierulf, Markus Enzenberger, Hiroshi Yamashita, Jimmy Lu, Sinichi Sei, Peter Woitke, David Fotland, Zhixing Chen, Mick Reiss, auteurs respectifs de SmartGo, NeuroGo, Aya, GoStar, Katsunari, GoAhead, Many Faces of Go, Goemate, Go4++, programmes adversaires de Indigo, sur la "ladder" ou pendant un tournoi. Je les apprécie pour leur compétence et leur passion commune pour la programmation du go.

Je remercie vivement tout les membres de l'équipe GNU Go, dirigée par Daniel Bump. Grâce à GNU Go, la programmation du go s'est ouverte aux communautés informatique, académique et aussi à celle des joueurs de go. Je remercie Jérome Dumonteil pour le transfert de mon travail de thèse vers l'équipe de GNU Go, à propos de la reconnaissance des territoires et pour la publicité de "l'algorithme 5-21" dans la communauté de la programmation du go qui en a découlé.

Je remercie Jean-Charles Pomerol, professeur d'informatique à l'université Paris 6, directeur du LAFORIA en 1997, directeur de la revue RIA, et Jean-Marc Labat, directeur du CRIP5, et animateur de la rubrique "Jeu" dans cette revue.

Je remercie les mathématiciens de mon UFR à Paris 5, en particulier Thierry Cabanal-Duvillard, maître de conférences, pour ses conseils en statistiques ou probabilités.

Je remercie Sylvie Després, maître de conférences en informatique à l'université Paris 5, habilitée à diriger des recherches, et Marie-Jo Caraty, à l'époque maître de conférences en informatique à l'université Paris 6 et désormais professeur en informatique à l'IUT de Paris 5, pour la force de conviction dont elles ont fait preuve à des moments clé de mon parcours.

Je remercie Martine Chauffeté, enseignante d'anglais au Centre Technique de Langues (CTL) de Paris 5, pour l'aide précieuse apportée dans la rédaction de mes publications en anglais.

Je remercie Thierry Raedersdorff et Laurent Moineau, ingénieurs système de l'UFR, pour leurs conseils d'utilisation des machines Linux de la pédagogie de l'UFR, inoccupées pendant le week-end ou la nuit.

Je remercie ma femme, Valérie Sion, pour son écoute et son soutien à ce travail. Je remercie mes enfants : Pablo, Lucas et Neïma pour leur fraîcheur d'esprit sans cesse renouvelée.

Enfin, je remercie infiniment mes parents, André et Anne-Marie, pour tout ce qu'ils m'ont apporté depuis toujours.

Références

[1] International Computer Game Association (ICGA). www.cs.unimaas.nl/icga/.

[2] Légifrance, le service public de l'accès au droit. www.legifrance.gouv.fr/.

[3] GdR I3 : Information Interaction Intelligence. sis.univ-tln.fr/gdri3/.

[4] Journées Nationales sur les Modèles de Raisonnement 99 (jnmr99). www.cril.univ-artois.fr/∼marquis/JNMR99/articlesJNMR-99.html, Mars 1999.

[5] 21st Century Cup. www.intelligentgo.org/en/igf/21cc2002/new-index.html, 2002.

[6] Computer Olympiad Graz 2003. www.cs.unimaas.nl/Olympiad2003/, 2003.

[7] B. Abramson. Expected-outcome : a general model of static evaluation. *IEEE Transactions on PAMI*, 12 :182–193, 1990.

[8] E. Agbodjan-Prince, Y. Aït Bachir, S. Bourham, S. Deleplace, and L. Sanchez-Garrido. Project Statistics and Programming, January 2004.

[9] V. Airault. Application des algorithmes génétiques au jeu d'Hex. Technical report, Université Paris 5, 2001. Rapport de stage de MIAIF2.

[10] L.V. Allis. *Searching for solutions in games and Artificial Intelligence*. PhD thesis, Vrije Universitat Amsterdam, 1994.

[11] L.V. Allis, M. van der Meulen, and H.J. van den Herik. Proof-number search. *Artificial Intelligence*, 66 :91–124, 1994.

[12] T. Anantharaman, M. Campbell, and F. Hsu. Singular extensions : adding selectivity to brute force searching. *Artificial Intelligence*, 43(1) :99–109, 1989.

[13] V. Anshelevich. Hexy. http ://home.earthlink.net/∼vanshel/.

[14] V. Anshelevich. A hierarchical approach to computer Hex. *Artificial Intelligence*, 134 :101–120, 2002.

[15] C. Mary B. Bouzy. Etat actuel de la programmation du jeu de Go. *Revue Flux*, (184) :38–42, Juin 1997. revue de l'amicale des ingénieurs de l'école supérieure d'électricité.

[16] E. Baum and W. Smith. A bayesian approach to relevance in game-playing. *Artificial Intelligence*, 97 :195–242, 1997.

[17] D. Beal. A generalised quiescence search algorithm. *Artificial Intelligence*, 43 :85–98, 1990.

[18] D. Benson. Life in the game of Go. *Information Sciences*, 10 :17–29, 1976.

[19] E. Berlekamp. Introductory overview of mathematical Go endgames. In *Proceedings of Symposia in Applied MAthematics*, volume 43, 1991.

[20] E. Berlekamp. *Games of No Chance*, chapter The Angel problem, pages 3–12. MSRI Publications, 1996.

[21] E. Berlekamp. *More Games of No Chance*, chapter The 4G4G4G4G4 problems and solutions, pages 231–242. MSRI Publications, 2002.

[22] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways*. Academic Press, Londres, 1982.

[23] E. Berlekamp and D. Wolfe. *mathematical go : chilling gets the last point*. AK Peters, 1994.

[24] H. Berliner. The B* Tree Search Algorithm : a best-first proof procedure. *Artificial Intelligence*, 12 :23–40, 1979.

[25] H. Bernery, B. Bonnenfant, B. Jezequel, S. Saidi, and S. Thiébaut. Project Statistics and Programming, January 2004.

[26] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134 :201–240, 2002.

[27] C. Bishop. *Neural networks and pattern recognition*. Oxford University Press, 1995.

[28] M. Boon. A pattern matcher for Goliath. *Computer Go*, 13 :13–23, 1990.

[29] M. Boon. Overzicht van de ontwikkeling van een spelend programma. Master's thesis, 1992.

[30] B. Bouzy. Explicitation de connaissances du joueur de Go. Technical report, LAFORIA, La Motte d'Aveillans, Mars 1994. Poster au Premier colloque jeunes chercheurs en sciences cognitives.

[31] B. Bouzy. Modélisation des groupes au jeu de Go. Technical report, LAFORIA, Marseille, Septembre 1994. Poster au 2ème colloque jeunes chercheurs en Intelligence Artificielle.

[32] B. Bouzy. Les ensemble flous au jeu de Go. In *Rencontres Francophones sur Logique Floue et Applications*, Paris, 1995.

[33] B. Bouzy. *Modélisation cognitive du joueur de Go*. PhD thesis, Université Paris 6, 1995.

[34] B. Bouzy. On Meta-game architectures useful in the game of Go. In *Workshop on Reflection and Meta-level architectures, IJCAI*, pages 80–85, Montréal, 1995.

[35] B. Bouzy. The Indigo program. In *2nd Game Programming Workshop in Japan*, pages 197–206, Hakone, 1995.

[36] B. Bouzy. Explicitation de connaissances non conscientes par modélisation computationnelle dans un domaine complexe. In *2ème Colloque Jeunes Chercheurs en Sciences Cognitives*, pages 276–279, Giens, 1996.

[37] B. Bouzy. Spatial Reasoning in the game of Go. In *Workshop on Representations and Processes in Vision and Natural Language, ECAI*, pages 78–80, Budapest, 1996.

[38] B. Bouzy. There are no winning moves except the last. In *6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 197–202, Grenade, 1996.

[39] B. Bouzy. Un modèle du jeu de go basé sur des interactions. In Joël Quinqueton J.P. Muller, editor, *IA distribuée et systèmes multi-agents*, pages 73–83, Port-Camargue, 1996. Hermes.

[40] B. Bouzy. Incremental updating of objects in indigo. In *4th Game Programming Workshop in Japan*, pages 55–64, Hakone, 1997.

[41] B. Bouzy. An interaction-based model for situated agents. In *International Conference on Multi-Agent Systems*, pages 399–400, Paris, 1998.

[42] B. Bouzy. Complex games in practice. In *5th Game Programming Workshop in Japan*, pages 53–60, Hakone, 1999.

[43] B. Bouzy. Alfa-Beta enhancements vs domain-dependent enhancements through a small Go board study. unpublished, 2001.

[44] B. Bouzy. Go patterns generated by retrograde analysis. In *Computer Olympiad Workshop*, Maastricht, 2001.

[45] B. Bouzy. Le role des concepts spatiaux dans la programmation du jeu de go. *Revue d'Intelligence Artificielle*, 15(2) :143–172, 2001.

[46] B. Bouzy. Indigo home page. www.math-info.univ-paris5.fr/∼bouzy/INDIGO.html, 2002.

[47] B. Bouzy. A small Go board Study of metric and dimensional Evaluation Functions. In Y. Björnsson J. Schaeffer, M. Müller, editor, *Computers and Games*, number 2883 in Lecture Notes in Computer Science, pages 376–392, Edmonton, 2002. Springer.

[48] B. Bouzy. Associating knowledge and Monte Carlo approaches within a go program. In *7th Joint Conference on Information Sciences*, pages 505–508, Raleigh, 2003.

[49] B. Bouzy. La conférence Computers and Games 2002. *Revue d'Intelligence Artificielle*, 17(4) :687–693, 2003.

[50] B. Bouzy. Mathematical morphology applied to computer go. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(2) :257–268, March 2003.

[51] B. Bouzy. The move decision process of Indigo. *International Computer Game Association Journal*, 26(1) :14–27, March 2003.

[52] B. Bouzy. Associating knowledge and Monte Carlo approaches within a Go program. *Information Sciences*, 2004. A paraître.

[53] B. Bouzy. Associating shallow and selective global tree search with Monte Carlo for 9x9 go. In *Proceedings of the fourth Computers and Games conference (CG04)*, 2004.

[54] B. Bouzy and T. Cazenave. Shared concepts between complex domains and the game of Go. In *International Conference on Context*, Rio de Janeiro, 1997.

[55] B. Bouzy and T. Cazenave. Computer Go : an AI oriented survey. *Artificial Intelligence*, 132 :39–103, 2001.

[56] B. Bouzy and B. Helmstetter. Monte Carlo Go Developments. In Ernst A. Heinz H. Jaap van den Herik, Hiroyuki Iida, editor, *10th Advances in Computer Games*, pages 159–174, Graz, 2003. Kluwer Academic Publishers.

[57] B. Bouzy and B. Victorri. Go et Intelligence Artificielle. *Bulletin de l'AFIA*, (10), Juillet 1992.

[58] M. Brooks. Go for it. *New Scientist*, pages 38–40, April 2002.

[59] B. Brügmann. Monte Carlo go. www.joy.ne.jp/welcome/igs/Go/computer/-mcgo.tex.Z, 1993.

[60] D. Bump. GNU Go home page. www.gnu.org/software/gnugo/devel.html, 2003.

[61] M. Buro. *Methods for the evaluation of game positions using examples.* PhD thesis, Paderborn University, 1994.

[62] M. Buro. ProbCut : an effective selective extension of the alpha-beta algorithm. *ICCA Journal*, 18(2) :71–76, 1995.

[63] M. Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence Journal*, 134 :85–99, 2002.

[64] R. Caillois. *Des jeux et des hommes.* Flammarion, 1948.

[65] M. Campbell, A.J. Hoane, and F-H Hsu. Deep Blue. *Artificial Intelligence*, 134 :57–83, 2002.

[66] M. Campbell and T. Marsland. A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20 :347–367, 1983.

[67] A-E. Carsin, A. Grand, M. Le Guen, C. Roy, and C. Saillé. Project Statistics and Programming : Amazons Game and Monte Carlo's method, January 2004.

[68] S. Castillo and J.F. Goudou. Programmation d'un logiciel de go. Méthode de Monte Carlo. Technical report, Ecole Nationale Supérieure des Télécommunications, 2003. Projets.

[69] T. Cazenave. Golois. www.ai.univ-paris8.fr/∼cazenave/Golois.html.

[70] T. Cazenave. *Système d'apprentissage par auto-observation. Application au jeu de Go.* PhD thesis, Université Paris 6, 1996.

[71] T. Cazenave. Abstract Proof Search. In I. Frank T. Marsland, editor, *Computers and Games*, number 2063 in Lecture Notes in Computer Science, pages 39–54. Springer, 2000.

[72] T. Cazenave. A generalized threats Search Algorithm. In Y. Björnsson J. Schaeffer, M. Müller, editor, *Computers and Games*, number 2883 in Lecture Notes in Computer Science, pages 75–87. Springer, 2002.

[73] T. Cazenave. Gradual Abstract Proof Search. *International Computer Game Association Journal*, 25(1) :3–15, March 2002.

[74] K. Chen. The move decision process of Go Intellect. *Computer Go*, 14 :9–17, 1990.

[75] K. Chen. Some practical techniques for global search in go. *ICGA Journal*, 23(2) :67–74, 2000.

[76] K. Chen. Knowledge and Search in Computer Go. In *6th Game Programming Workshop in Japan*, Hakone, 2001.

[77] K. Chen. A study of decision error in selective game tree search. *Information Sciences*, 135 :177–186, 2001.

[78] K. Chen. GNU Go wins 19x19 Go Tournament. *International Computer Game Association Journal*, 26(4) :261–262, December 2003.

[79] K. Chen and Z. Chen. Static analysis of life and death in the game of Go. *Information Sciences*, pages 113–134, 1999.

[80] A. Cohn. Qualitative spatial representation and reasoning techniques. In *Proceedings KI 91*, volume 1303 of *lecture Notes in Artificial Intelligence*, pages 1–30. Springer, 1997.

[81] J. Conway. *On Numbers and Games*. Academic Press, 1976.

[82] F. Dahl. Honte, a go-playing program using neural nets. In J. Fürnkranz and M. Kubat, editors, *Workshop at the 16th International Conference on Machine Learning (ICML-99)*, Bled, Slovenia, June 1999.

[83] C. Donninger. Null move and deep search : selective search heuristics for obtuse chess programs. *ICCA Journal*, 16(3) :137–143, 1993.

[84] M. Egenhofer. Reasoning about Binary Topological Relations. In *Proceedings Advance in Spatial Databases*, volume 525 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 1991.

[85] M. Enzenberger. Computer Go Bibliography. www.markus-enzenberger.de/.

[86] M. Enzenberger. NeuroGo. www.markus-enzenberger.de/neurogo.html.

[87] M. Enzenberger. The integration of a priori knowledge into a go playing program. www.markus-enzenberger.de/, 1996.

[88] M. Enzenberger. Evaluation in Go by a Neural Network using soft segmentation. In Ernst A. Heinz H. Jaap van den Herik, Hiroyuki Iida, editor, *10th Advances in Computer Games*, pages 97–108, Graz, 2003. Kluwer Academic Publishers.

[89] D. Fotland. The Many Faces of Go. www.smart-games.com/manyfaces.html.

[90] D. Fotland. The program G2. *Computer Go*, 1 :10–16, 1986.

[91] D. Fotland. An algorithm for simple ladders. *Computer Go*, 2 :5–7, 1987.

[92] D. Fotland. Static Eye in "The Many Faces of Go". *ICGA Journal vol 25 n4*, pages 203–210, December 2002.

[93] A. Fraenkel. *Games of No Chance*, volume 29, chapter What is a game ?, pages 43–60. MSRI Publications, 1996.

[94] A. Fraenkel and D. Lichtenstein. Computing a perfect strategy for N by N chess requires time exponential. *Journal of Combinatorial Theory, serie A*, 31(2) :199–214, 1981.

[95] R.D. Greenblatt, D.E. Eastlake, and S.D. Crocker. The Greenblatt chess program. In *Fall Joint Computing Conference*, volume 31, pages 801–810, New york ACM, 1967.

[96] A. Junghanns. Are there practical alternatives to Alpha-Beta ? *ICCA Journal*, 21(1) :14–32, March 1998.

[97] A. Kierulf. SmartGo. www.smartgo.com/.

[98] A. Kierulf, K. Chen, and J. Nievergelt. Smart Game Board and Go Explorer : a Study in software and knowledge engineering. *Communications of the ACM*, 33(2) :152–166, February 1990.

[99] D. Knuth and R. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6 :293–326, 1975.

[100] R. Korf. Depth-first Iterative Deepening : an Optimal Admissible Tree Search. *Artificial Intelligence*, 27 :97–109, 1985.

[101] R. Korf and D. Chickering. Best-first search. *Artificial Intelligence*, 84 :299–337, 1994.

[102] B. Kuipers. Modeling Spatial Knowledge. *Cognitive Science*, 2 :129–153, 1978.

[103] H. Landman. *Games of No Chance*, volume 29, chapter Eyespace Values in Go, pages 227–258. MSRI Publications, 1996.

[104] D. Lichtenstein and M. Sipser. Go is polynomial space hard. *Journal of the ACM*, 2 :393–401, 1980.

[105] J. Lieberum. Amazong. web2.whosting.ch :1412/jens-lieb/amazong/amazong.html.

[106] J. Lieberum. An Evaluation Function in the Game of Amazons. In Ernst A. Heinz H. Jaap van den Herik, Hiroyuki Iida, editor, *10th Advances in Computer Games*, pages 299–308, Graz, 2003. Kluwer Academic Publishers.

[107] S. Lucas and G. Kendall. IEEE Symposium on Computational Intelligence in Games. cswww.essex.ac.uk/Research/cigames/, April 2005.

[108] K. Marignac, J. Petit, M. Ribeiro, and G. Sprauel. Project Statistics and Programming, January 2004.

[109] T. Marsland. A review of game-tree pruning. *ICCA Journal*, 9(1) :3–19, 1986.

[110] G. Martin. *More Games of No Chance*, volume 42, chapter Restoring Fairness to DukeGo, pages 79–88. MSRI Publications, 2002.

[111] D. McAllester. Conspiracy Numbers for Min-Max Search. *Artificial Intelligence*, 35 :287–310, 1988.

[112] R. Moneret. *Stratégos : un système multi-jeux utilisant la théorie combinatoire des jeux, capable d'apprendre automatiquement les dépendances entre sous-jeux locaux*. PhD thesis, Université Paris 6, 2000.

[113] M. Müller. Explorer. web.cs.ualberta.ca/∼mmueller/cgo/explorer.html.

[114] M. Müller. *Computer Go as sum of local games : an application of combinatorial game theory*. PhD thesis, Swiss Federal Institute of Technology, Zürich, 1995.

[115] M. Müller. Decomposition Search : A Combinatorial Games Approach to Game Tree Search, with Applications to Solving Go Endgame. In *IJCAI*, pages 578–583, 1999.

[116] M. Müller. Global and local game tree search. *Information Sciences*, 135 :187–206, 2001.

[117] M. Müller. Computer Go. *Artificial Intelligence*, 134 :145–179, 2002.

[118] M. Müller and R. Gasser. *Games of No Chance*, volume 29, chapter Experiments in Computer Go endgames, pages 273–284. MSRI Publications, 1996.

[119] M. Müller and T. Tegos. *More Games of No Chance*, volume 42, chapter Experiments in Computer Amazons, pages 243–260. MSRI Publications, 2002.

[120] A.J. Palay. *Searching with probabilities*. Morgan Kaufman, 1985.

[121] J. Pearl. SCOUT : a simple game-searching algorithm with proven optimal properties. pages 143–145.

[122] J. Pearl. Asymptotic properties of minimax trees and game-searching procedures. *Artificial Intelligence*, 14 :113–138, 1980.

[123] J. Pitrat. *Métaconnaissance, futur de l'intelligence artificielle*. Hermès, 1990.

[124] A. Plaat, J. Schaeffer, W. Pils, and A. de Bruin. Best-first fixed depth minimax algorithms. *Artificial Intelligence*, 87 :255–293, November 1996.

[125] M. Reiss. Go++. www.goplusplus.com/.

[126] M. Reiss. Mick's Computer Go Page. www.reiss.demon.co.uk/webgo/compgo.htm.

[127] P. Ricaud. *Gobelin : une approche pragmatique de l'abstraction appliquée à la modélisation de la stratégie élémentaire au jeu de go*. PhD thesis, Université Paris 6, 1996.

[128] P. Ricaud. A model of strategy for the game of go using abstraction mechanisms. In *Proceedings IJCAI*, pages 678–683, Nagoya, Japan, 1997.

[129] R. Rivest. Game-tree searching by min-max approximation. *Artificial Intelligence*, 34(1) :77–96, 1988.

[130] A. Sadikov, I. Bratko, and I. Kononenko. Search versus Knowledge : an empirical study of minimax on KRK. In Ernst A. Heinz H. Jaap van den Herik, Hiroyuki Iida, editor, *10th Advances in Computer Games*, pages 33–44, Graz, 2003. Kluwer Academic Publishers.

[131] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3 :210–229, 1959.

[132] A. Samuel. Some studies in machine learning using the game of checkers : Recent progress. *IBM Journal of Research and Development*, 11 :601–617, 1967.

[133] J. Schaeffer. The history heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transcations on Pattern Analysis and Machine Intelligence*, 11(11) :1203–1212, November 1989.

[134] J. Schaeffer. Conspiracy Numbers. *Artificial Intelligence*, 43 :67–84, 1990.

[135] J. Schaeffer. *One Jump Ahead : Challenging Human Supremacy in Checkers*. Springer-Verlag, 1997.

[136] J. Schaeffer. The Games Computers (and People) Play. *AAAI*, 2000.

[137] J. Schaeffer and R. Lake. *Games of No Chance*, volume 29, chapter Solving the game of Checkers, pages 119–133. MSRI Publications, 1996.

[138] J. Schaeffer, A. Plaat, and A. Junghanns. Unifying single-agent and two-player search. *Information Sciences*, 135 :151–175, 2001.

[139] J. Schaeffer and J. van den Herik. Games, Computers, and Artificial Intelligence. *Artificial Intelligence*, 134 :1–7, 2002.

[140] N. Schraudolf, N. Dayan, and T. Sejnowski. Temporal Difference learning of a position evaluation in the game of Go. In Cowan, Tesauro, and Alspector, editors, *Neural Information Processing Systems*, volume 6, pages 817–824. Morgan Kaufmann, San Francisco, 1994.

[141] S. Sei and T. Kawashima. A solution of go on 4x4 board by game tree search program. In *4th game Informatics Group Meeting in Japan*, pages 69–76, 2000. in Japanese.

[142] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[143] C.E. Shannon. Programming a computer to play Chess. *Philosoph. Magazine*, 41 :256–275, 1950.

[144] B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134 :241–275, 2002.

[145] D.J. Slate and L.R. Atkin. Chess 4.5 - the northwestern university chess program. In P. Frey, editor, *Chess Skill in Man and Machine*, pages 82–118. Springer-Verlag, 1977.

[146] R. Snatzke. *More Games of No Chance*, volume 42, chapter Exhaustive Search in Amazons, pages 261–278. MSRI Publications, 2002.

[147] B. Spight. *More Games of No Chance*, chapter Go Thermography : the 4/21/98 Jiang-Rui Endgame, pages 89–106. MSRI Publications, 2002.

[148] G.C. Stockman. A minimax algorithm better than Alpha-Beta ? *Artificial Intelligence*, 12 :179–196, 1979.

[149] R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3 :9–44, 1988.

[150] R. Sutton and A. Barto. *Reinforcement Learning : an introduction*. MIT Press, 1998.

[151] M. Tajima and N. Sanechika. Estimating the Possible Ommission Number for Groups in Go by the Number of n-th Dame. In J. van den Herik and H. Iida, editors, *Proceedings of the First International Conference on Computers and Games*, volume 1558 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 1998.

[152] T. Tegos. The Game of Amazons. web.cs.ualberta.ca/∼tegos/amazons/.

[153] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38 :58–68, 1995.

[154] G. Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134 :181–199, 2002.

[155] G. Tesauro and G. Galperin. On-line policy improvement using Monte Carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074, Cambridge MA, 1996. MIT Press.

[156] T. Thomsen. Lambda-search in game tress, with application to go. *International Computer Game Association Journal*, 23(4) :203–217, December 2000.

[157] K. Thomson. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3) :131–139, 1986.

[158] K. Thomson. 6-piece endgames. *ICCA Journal*, 19 :215–226, 1996.

[159] E. Thorpe and W. Walden. A partial analysis of go. *Computer Journal*, 7(3) :203–207, 1964.

[160] E. Thorpe and W. Walden. A computer assisted study of go on MxN boards. *Information Sciences*, 4 :1–33, 1972.

[161] Erika Valencia, Gérard Ligozat, and Jean-Paul Sansonnet. Séminaire quando et urbi. www.limsi.fr/Individu/erika/QetU.html.

[162] H.J. van den Herik, J.W.H.M. Uiterwijk, and J. van Rijswijck. Games solved : Now and in the future. *Artificial Intelligence*, 134 :277–311, 2002.

[163] E. van der Werf. Aya wins 9x9 Go Tournament. *International Computer Game Association Journal*, 26(4) :263, December 2003.

[164] E. van der Werf, H.J. van den Herik, and J.W.H.M. Uiterwijk. Solving Go on Small Boards. *International Computer Game Association Journal*, 26(2) :92–107, June 2003.

[165] J. van Rijswijck. Computer Hex : Are bees better than fruitflies ? Master's thesis, Alberta University, 2000.

[166] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[167] C. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

[168] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8 :279–292, 1992.

[169] N. Wedd. Goemate wins Go Tournament. *International Computer Game Association Journal*, 23(3) :175–177, September 2000.

[170] P. Woitke. Goahead. astro.physik.tu-berlin.de/∼woitke/GoAhead.html.

[171] T. Wolf. The program GoTools and its computer-generated tsume-go database. In *1st Game Programming Workshop in Japan*, pages 84–91, 1994.

[172] T. Wolf. About problems in generalizing a tsumego program to open positions. In *3rd Game Programming Workshop in Japan*, pages 20–26, 1996.

[173] T. Wolf. Forward pruning and other heuristic search techniques in tsume go. *Information Sciences*, 122 :59–76, 2000.

[174] H. Yamashita. Hiroshi's Computer Shogi and Go. www32.ocn.ne.jp/∼yss/index.html.

[175] A. Zobrist. A model of visual organization for the game of go. In *Proceedings AFIPS Spring Joint Computer Conference*, pages 103–111, Boston, 1969. AFIP Press.

[176] A. Zobrist. A new hashing method with application for game playing. *ICCA Journal*, 13(2) :69–73, 1990.