REVUE D'INTELLIGENCE ARTIFICIELLE

SOMMAIRE • VOLUME 31 – N° 3/2017

IA DES JEUX INFORMATISÉS

INTRODUCTION

Parmi les nombreuses applications possibles de l'intelligence artificielle, ou IA, les jeux informatisés constituent un terrain d'expérimentation pratique et efficace pour le développement de méthodes et techniques innovantes. Par jeu informatisé, nous comprenons tous jeux pouvant être joués sur un ordinateur ou une machine : jeux vidéo, jeux plateau comme les échecs ou le go, jeux sérieux, etc. Outre le côté ludique, les jeux informatisés proposent des environnements simplifiés par rapport au monde réel, mais aux règles toutefois suffisamment riches pour poser des problèmes scientifiquement pertinents : compréhension d'un environnement dynamique, extraction de connaissances, prise de décision à partir d'informations souvent incomplètes, planification d'actions, recherche de chemin, coopération, apprentissage de comportement crédible souhaité, optimisation, etc. Ces problèmes sont communs avec d'autres domaines applicatifs tels que la robotique, mais les jeux informatisés offrent une plus grande flexibilité : grâce à leur caractère dématérialisé, ce sont des environnements peu chers, transportables, sans besoin de maintenance, et où l'écoulement du temps peut être accéléré à loisir.

Ces dernières années, nous voyons se développer dans la recherche académique et industrielle une grande accélération des avancées en IA où les jeux constituent un environnement d'étude privilégié : citons AlphaGo de Google DeepMind qui domine maintenant les grands maîtres au jeu de Go en réussissant une habile coopération entre méthodes de *Monte Carlo Tree Search* et de *Deep Reinforcement Learning* ; citons aussi DeepStack et Libratus qui battent des professionels au poker, jeu où le caractère incomplet de l'information rend très difficile la conception d'un agent autonome efficace. Cette effervescence de l'IA autour des jeux est confirmée par le nombre croissant de conférences académiques accueillant des compétitions d'IA autour de jeux, mais aussi par l'explosion de plate-formes disponibles pour développer des agents autonomes dans des jeux. Ces plate-formes sont bien souvent développées par de grands groupes dont le jeu n'est pourtant pas le cœur de métier, tel que Minecraft Malmö (Microsoft Research), Universe (Open AI), TorchCraft (Facebook), et bientôt une API pour StarCraft 2 (Blizzard Entertainment - Google DeepMind).

Ce numéro spécial de la *Revue d'Intelligence Artificielle* expose des travaux et directions de recherche en cours dans le domaine de l'IA des jeux informatisés, et met en perspective les enjeux auxquels cette recherche sera confrontée dans les années à venir. Nous avons reçu 9 soumissions et retenu 4 contributions après relecture, balayant un large spectre de ce qui peut être fait en IA des jeux.

Après avoir exposé l'état de l'art des IA de bridge, la première contribution présente les travaux effectués autour de la recherche de meilleures graines aléatoires dans le cadre du bridge. La deuxième contribution propose un algorithme innovant, MAC-UCB, pour résoudre des problèmes de satisfaction de contraintes stochastiques modélisant un grand nombre de jeux de plateau. Ces deux contributions présentent des travaux aboutissant à des agents autonomes, Wbridge5

et WoodStock, remportant les championnats mondiaux d'IA dans leur domaine respectif, à savoir le bridge et la compétition General Game Playing. La troisième contribution s'intéresse à la planification d'actions dans un très grand espace de décision. Par l'utilisation de traces d'exécutions et l'estimation de l'*utilité* des solutions contenues dans ces traces, les auteurs de cet article proposent un moyen de réduire efficacement l'espace de recherche, et appliquent leur méthode à un jeu Tamagotchi. Enfin, la quatrième contribution propose elle aussi une méthode simple et élégante de réduction d'espace de recherche pour des solveurs de problème de satisfaction de contraintes quantifiées dans le cadre de jeux à deux joueurs à horizon fini et information complète, aboutissant à un solveur nommé QuaCode surpassant les solveurs de l'état de l'art.

Nous tenons à remercier vivement les relecteurs qui ont contribué à l'élaboration de ce numéro spécial. Nous remercions également Yves Demazeau, rédacteur en chef de la *Revue d'Intelligence Artificielle*, qui a soutenu ce projet depuis le début.

FLORIAN RICHOUX
Université de Nantes, LS2N

CAROLE ADAM
Université Grenoble-Alpes, LIG

CÉDRIC BUCHE
École Nationale d'Ingénieurs de Brest, Lab-STICC

TRISTAN CAZENAVE
Université Paris-Dauphine, LAMSADE

COMITÉ DE LECTURE DE CE NUMÉRO

Le bridge, nouveau défi de l'intelligence artificielle ?

Véronique Ventos, Olivier Teytaud

LRI, INRIA, Univ Paris-Sud, CNRS, Université Paris-Saclay, France
ventos@lri.fr,olivier.teytaud@inria.fr

RÉSUMÉ. *Le jeu de bridge est un jeu extrêmement complexe et ce aussi bien pour les humains que pour les programmes de bridge. Contrairement au jeu de Go où l'IA AlphaGo a récemment atteint le niveau de grand maître, les IA de bridge sont encore loin des meilleurs joueurs humains. La différence essentielle entre les deux jeux porte sur l'aspect information incomplète omniprésent dans le bridge. Un programme expert de ce jeu pourrait donc réaliser des tâches différentes et complémentaires de celles traitées par AlphaGo. La première partie de cet article est consacrée à la présentation des différents aspects du bridge et de quelques pistes de réflexion sur les différents challenges qui lui sont inhérents ainsi que des résultats obtenus par la communauté bridge informatisé. Dans la seconde partie, nous présentons nos travaux concernant l'adaptation au bridge d'une méthode récente permettant d'optimiser des IA de jeux en recherchant une graine aléatoire meilleure que les autres sur le jeu concerné. L'IA Wbridge5 développée par Yves Costel a été boostée avec la meilleure graine trouvée a l'issue de ces expériences et a remporté les championnats du monde de robots de bridge en septembre 2016.*

ABSTRACT. *The game of bridge is very complex both for humans and for computer bridge programs. While the AI Go AlphaGo recently reached the level of grandmaster go, it is not the case for bridge where robots are still far from the best human players. The main difference between the two games is that bridge is a partially observable game. An expert program on bridge could therefore perform tasks that are different from and complementary to those handled by AlphaGo. The first part of this article is devoted to the presentation of various aspects of the bridge and some thoughts on the numerous challenges inherent to them together with a state of art of the computer bridge community. In the second part, we present our work related to the adaptation to bridge of a recent method using for boosting game AIs by seeking a random seed better than the others on a particular game. The AI bridge Wbridge5 developed by Yves Costel has been boosted with the best seed found on the outcome of these experiments and won the world championship bridge robots in September 2016.*

MOTS-CLÉS : *bridge informatisé, apprentissage automatique, monte-carlo, amélioration d'IA.*

KEYWORDS: *computer bridge, machine learning, monte-carlo, boosting AI.*

DOI:10.3166/RIA.31.249-279

1. Introduction

« Le bridge est un jeu de cartes, de type jeu de levées, consistant à comptabiliser le nombre de levées réalisées. Il se pratique avec un jeu de 52 cartes, par quatre joueurs, opposant deux équipes composées de deux partenaires » (wikipédia).
Si l'on s'en tient à cette définition, le bridge paraît bien inoffensif, pourtant nous voulons proposer ce jeu comme nouveau challenge de l'IA. En effet, le bridge est un jeu extrêmement complexe et ce aussi bien pour les humains que pour les robots de bridge qui ne dépassent pas le niveau de bons amateurs.

La conception d'un robot joueur de go de niveau grand maître a été pendant des années un défi majeur pour les chercheurs en intelligence artificielle. Le nombre très élevé de combinaisons (de l'ordre de 10^{600}) rend impossible l'utilisation de méthodes traditionnelles de type *force brute* efficaces sur d'autres jeux comme les échecs ou les dames anglaises. Ceux-ci ont bénéficié des progrès liés à la puissance des ordinateurs qui ont permis la conception de *superordinateurs* pouvant évaluer un nombre gigantesque de positions par seconde. Le plus connu est le superordinateur *Deep Blue* développé par IBM qui a battu en 1997 le champion du monde d'échecs de l'époque Garry Kasparov. Cependant, c'est le programme de dames anglaises *Chinook* qui est le premier à avoir obtenu un titre de champion du monde contre un humain en 1994. Dans le cas des dames anglaises, l'approche basée sur une expertise humaine [1] a conduit à la résolution complète du jeu avec la définition d'une stratégie de jeu infaillible et une preuve publiée dans (Schaeffer *et al.*, 2007) que si les deux adversaires jouent parfaitement le meilleur résultat possible sera une partie nulle. Les dames anglaises est donc un jeu résolu ce qui n'est pas le cas des échecs même si le superordinateur d'échecs Hydra (200 millions de positions évaluées par seconde) a obtenu un classement jamais atteint par les humains. Le classement est basé sur le calcul de points Elo utilisés pour comparer les performances des joueurs. Au premier janvier 2017, onze joueurs seulement ont dépassé 2 800 points dans leur carrière sans jamais atteindre 2900. Hydra a obtenu près de 3000 points Elo et les programmes d'échecs plus récents comme Stockfish, Houdini ou Komodo ont dépassé 3200 Elo.

En ce qui concerne les programmes de Go, le facteur de branchement de l'arbre de recherche et la difficulté croissante pour évaluer une position rendent inefficaces les programmes de type force brute. Pour améliorer l'évaluation des positions, les programmes ont commencé à utiliser à partir de 1993 des méthodes de Monte-Carlo dont le principe a été inventé par Nicholas Metropolis et publié dans (Metropolis, Ulam, 1949). En dépit de cette amélioration et de la puissance de calcul des ordinateurs de l'époque, jusqu'en 2006 les meilleurs programmes ne dépassent pas le niveau d'un joueur amateur moyen. La première révolution est arrivée avec l'introduction de stratégies couplant l'utilisation de méthodes de Monte-Carlo et une adaptation au Go de l'algorithme UCT (*Upper Confidence bounds applied to Trees*) qui permet d'explorer plus souvent le meilleur coup dans un arbre de recherche. Les programmes Mogo

1. Les connaissances ont toutes été entrées manuellement dans le programme.

(projet TAO de l'INRIA Futurs et de l'école Polytechnique (Munos, Teytaud, 2006) et (Gelly *et al*., 2012)) et Crazystone (projet SEQUEL de l'Inria Futurs (Coulom, 2006)) ont utilisé cette nouvelle approche dès 2006, ce qui leur a permis de remporter des victoires contre des joueurs professionnels mais sur des plateaux plus petits que le goban classique (19X19) ou avec des pierres de handicap. En janvier 2016, une équipe de DeepMind, entreprise britannique spécialisée dans l'intelligence artificielle et les jeux, publie un article (Silver *et al*., 2016) sur la présentation d'AlphaGo le premier programme de Go ayant battu un joueur professionnel sur un goban classique sans handicap. AlphaGo est un programme utilisant et intégrant différentes techniques d'apprentissage relativement récentes. Deux mois plus tard, l'IA AlphaGo devient grand maître après sa victoire par quatre parties à une dans un match l'oppposant à Lee Sedol l'un des tout meilleurs joueurs de go au monde.

AlphaGo repose sur trois techniques : les réseaux de neurones profonds, les techniques de recherche de type monte-carlo et l'apprentissage renforcé ou RL pour Reinforcement Learning (Sutton, Barto, 1998). Deux réseaux profonds cohabitent, la tâche du premier est de prédire le prochain coup en utilisant les données de dizaines de milliers de parties d'excellents joueurs humains. Cette première étape d'imitation de l'humain permet à l'IA d'atteindre un niveau correct de jeu. Une fois ce niveau atteint, on fait jouer plusieurs millions de parties opposant l'IA à elle-même. Les nouvelles données générées durant cette étape serviront d'entrée au second réseau profond dont le but sera de renvoyer une évaluation d'un mouvement relativement à l'état courant du plateau. Des techniques d'apprentissage renforcé sont ensuite utilisées pour combiner les deux réseaux. La place de l'apprentissage automatique dans le processus est très importante ce qui est radicalement différent de l'approche utilisée par exemple par *Chinook* où aucune connaissance n'a été apprise par une méthode d'intelligence artificielle. A la suite de la victoire d'AlphaGo, beaucoup se sont demandé quel pourrait être le prochain défi de l'IA, le but étant d'étendre les champs d'application possibles en proposant la résolution de problèmes jusque-là non traités par un système d'IA. Un rapport récent (Heinrich, Silver, 2016) présente une application de Deep RL sur deux variantes du poker. Le poker est un jeu en information incomplète avec un aspect psychologique important. De plus, contrairement au jeu de go, le poker fait intervenir plus de deux joueurs. Dans la liste des prétendants figure aussi StarCraft, un jeu vidéo en temps réel où les joueurs n'ont pas une connaissance parfaite de leur environnement.

Nous pensons que définir un robot de bridge pouvant rivaliser avec les champions humains pourrait constituer une nouvelle étape pour l'intelligence artificielle. Les principales difficultés dans le cadre de ce jeu sont liées aux interactions entre les différentes étapes du jeu et au fait que les raisonnements sont réalisés en information incomplète. La diversité des modes de raisonnement nécessaires pour atteindre, voire dépasser, le niveau des meilleurs joueurs devrait conduire à l'élaboration d'une architecture hybride utilisant des méthodes liées aussi bien à la communauté computer game qu'à celle de l'apprentissage automatique numérique et symbolique. La définition même de cette architecture est un défi. En effet, il faut déterminer pour chaque tâche quelles sont les méthodes existantes les plus appropriées, étudier la faisabilité de leur adaptation au bridge, définir de nouvelles méthodes et enfin déterminer comment

intégrer les différentes parties du système. Cette architecture devra comprendre une adaptation des méthodes utilisées par AlphaGo.

Les avantages pour utiliser le bridge comme application sont les suivants :

– Les règles sont simples et connues de tous.

– Le jeu est séquentiel et facilement modélisable.

– Le score est incrémental, le bridge ne fait donc pas partie des jeux où l'issue est incertaine jusqu'à la fin de la partie.

– Il existe un important volume de données relatives à des parties jouées par des humains ou des robots dans un format standard. Un expert du bridge sachant interpréter les données peut effectuer un travail de classement suivant les tâches attendues.

– La chance intervient mais elle est gommée par la marque duplicate* [2] qui permet de faire jouer les mêmes donnes à des joueurs différents.

– Il n'y a pas de problèmes liés au temps réel.

La communauté bridge informatisé (ou computer bridge) est active depuis une quarantaine d'années mais le niveau des robots est loin des meilleurs joueurs humains comme c'était le cas pour le Go il y a quelques années. Les principales difficultés inhérentes au bridge sont relatives aux éléments suivants :

– C'est un jeu à information incomplète.

– Le bridge se joue à quatre (deux équipes de deux).

– Il y a une nécessité d'expliciter les raisonnements que l'on fait.

– Des facteurs psychologiques interviennent.

– L'expertise humaine ne peut suffire à résoudre le jeu.

– L'adaptation à un partenaire est longue et essentielle, elle nécessite une phase d'apprentissage y compris chez les champions.

Une partie de ces difficultés semble difficilement traitable en n'utilisant que les méthodes utilisées par AlphaGo, une IA experte du bridge pourrait donc réaliser des tâches différentes et complémentaires de celles traitées par AlphaGo.

Un des objectifs de cet article est de familiariser le lecteur aux différents aspects du bridge et de faire ressortir des problématiques qui pourront être résolues avec plusieurs techniques existantes ou à développer. C'est l'objet de la première partie de cet article. Plus précisément, la section 2 présente les bases du bridge et de ses différentes phases de jeu ainsi que les raisonnements qu'un joueur humain doit mettre en place pour résoudre les différents problèmes auxquels il est confronté. Les caractéristiques du bridge par rapport aux notions de la théorie des jeux sont décrites et commentées dans la section 3 dédiée au bridge informatisé avec un état de l'art et un bilan des résultats obtenus par cette communauté. La section 4 présente un ensemble de challenges à relever pour améliorer les programmes de bridge.

2. Un glossaire est donné en annexe A, la première occurrence des mots y figurant est suivie d'un astérisque.

La deuxième partie de l'article est dédiée à la présentation de nos travaux en cours et futurs sur le sujet. Le bridge étant à information incomplète les approches les plus fréquentes utilisent des simulations de Monte-Carlo. Ces simulations utilisent des générateurs de nombres pseudo-aléatoires dépendants de la graine utilisée. Dans la section 5, nous présentons nos travaux concernant l'adaptation au bridge d'une méthode récente permettant d'optimiser des IA de jeux en recherchant une graine aléatoire meilleure que les autres sur le jeu concerné (St-Pierre, Teytaud, 2014 ; Cazenave *et al*., 2015). L'IA de bridge Wbridge5 d'Yves Costel boostée avec la meilleure graine trouvée a l'issue de ces expériences a participé aux derniers championnats du monde des robots de bridge et les a remportés. La dernière section décrit brièvement nos travaux en cours sur la définition d'un cadre d'apprentissage supervisé visant à améliorer la prise de décisions des robots en fonction du contexte et donne un aperçu des perspectives de recherche qui sont nombreuses.

2. Le bridge

Nous présentons les bases du bridge dans cette section. Pour plus de détails, il existe des tutoriels interactifs en ligne [3] permettant de se familiariser avec le jeu.

2.1. Description du bridge

Le bridge est un jeu de 52 cartes qui oppose deux paires de joueurs. Les cartes sont distribuées aléatoirement aux quatre joueurs qui ne voient que leur *main** c'est à dire les treize cartes de leur jeu. Les règles sont connues des joueurs et sont relativement simples. Malgré ceci et en raison de l'aspect information incomplète le bridge est un jeu très compliqué et ce aussi bien pour les humains que pour les programmes de bridge. Le bridge comprend deux phases bien distinctes : les enchères et le jeu de la carte. Les raisonnements effectués lors de la seconde phase utilisent les informations hypothétiques dévoilées lors de la phase d'enchères.

Assimiler les notions de base prend en général plusieurs années aux humains et les méthodes d'apprentissage par exemple de l'Université du Bridge [4] préconisent de démarrer par une version bridée du jeu appelée mini-bridge où seul le jeu de la carte est réalisé.

2.1.1. La phase d'enchères

En bref : à l'issue des enchères, une des paires aboutit à un contrat final qui détermine l'éventuelle couleur d'atout* (il peut ne pas y en avoir, on parle alors de contrat à sans-atout*) et le nombre de levées* (plis) minimal que la paire s'engage à réaliser. Cette phase nécessite un investissement conséquent de la part d'un débutant pour apprendre

3. http://www.decouvertedubridge.com, https://en.wikipedia.org/wiki/Contract_bridge.

4. Département de la Fédération Française de Bridge (FFB) dirigé par Jean-Pierre Desmoulins et dédié à la jeunesse, la pédagogie et la recherche.

la signification des enchères par rapport à un système donné et surtout pour savoir l'utiliser à bon escient. Les enchères peuvent être vues comme un langage codé qu'on utilise pour faire passer des informations au partenaire sur sa main (principalement la distribution* et le nombre de points).

Plus précisément, le but de cette phase pour chaque camp est d'aboutir au meilleur contrat pour lui. Nous ne détaillons pas ici le calcul du score mais il faut savoir qu'il joue un rôle important dans le choix des enchères qu'un joueur peut faire. A partir d'un certain nombre de plis le camp obtient une prime si le contrat est gagné. Il existe des primes de *manche* liées aux contrats qui exigent un nombre minimum de levées dépendant de la couleur d'atout (9 plis à Sans Atout, 10 plis pour un contrat majeur* et 11 plis pour un contrat mineur*) et des primes *de chelems* sur les contrats où l'on s'engage à faire 12 plis ou tous les plis (*petit ou grand chelem*). Une décision fréquente pour un joueur concerne donc la hauteur du contrat : faut-il ou non demander un contrat en dessous du palier de la manche ou du chelem qu'on aura plus de chance de gagner puisqu'il exige un nombre de plis inférieurs mais qui rapportera moins ? Des études statistiques ont permis de déterminer des zones de points pour lesquelles il y a une probabilité assez grande de gagner les différents types de contrat. Pour prendre ces décisions, le joueur compte les points de sa main (cf annexe B) et essaie de transmettre cette information en utilisant des enchères. Les systèmes d'enchères qui varient d'une paire à l'autre tiennent compte de ces études et sont constituées de règles. Le score ne concerne que le nombre de plis réalisés et non le nombre de points dans les plis comme c'est le cas dans d'autres jeux de cartes. Les points associés aux cartes ne servent qu' à l'évaluation de la main et sont basés sur le fait qu'on a plus de chances de remporter un pli avec une carte forte qu'avec une petite carte.

Tâches à réaliser par un joueur humain

Un joueur doit à la fois produire une enchère parmi d'autres en fonction de sa main et de la séquence d'enchères courante et comprendre les enchères de son partenaire et de ses adversaires pour mettre à jour ses croyances. Un joueur humain va se trouver confronté à trois problèmes principaux :

1. La mémoire : le joueur doit se souvenir du système (il existe des systèmes plus ou moins complexes mais aucun n'est simple).

2. Le raisonnement : il doit raisonner afin de déduire des informations sur la main du partenaire mais aussi sur celles des adversaires puisque ceux-ci enchérissent aussi.

3. Le jugement : le joueur doit évaluer sa main en dehors des points et ce en fonction de toutes les informations dont il dispose. Ce jugement se modifie au cours de la séquence.

Pour un programme de bridge la partie mémoire ne posera pas de problème. Les parties raisonnement et jugement de main sont en revanche des challenges de taille pour les IA de bridge.

2.1.2. Le jeu de la carte

En bref : alors que le jeu de bridge est généralement considéré comme le plus complexe des jeux de cartes, ses règles sont relativement simples. Chaque joueur pose une carte à tour de rôle pour chaque pli. Pour cela un joueur doit fournir une carte de la couleur demandée quand il lui en reste et fournir n'importe quelle carte quand il n'a pas plus de carte dans cette couleur. Lorsque les quatre joueurs ont posé une carte, le pli se termine et est remporté par le joueur ayant posé la plus grosse carte dans la couleur demandée ou le plus gros atout. C'est lui qui posera la première carte du pli suivant. La donne se termine lorsque les cinquante-deux cartes ont été jouées. On compte le nombre de levées remportées par chaque camp (cette fois peu importe quel joueur de la paire a remporté le pli). Le score de la donne est calculé à partir de ce nombre et du nombre de levées requises par le contrat.

Après la phase des annonces*, on distingue le camp du déclarant (le joueur qui a fait la dernière enchère) et l'autre camp qu'on appelle « le camp de la défense » ou encore « le flanc ». Le défenseur à la gauche du déclarant va *entamer** c'est à dire choisir et poser une carte de son jeu sur la table. Après l'entame le jeu du partenaire du déclarant est étalé sur la table, il ne reste donc pour chaque joueur que 25 cartes inconnues à répartir entre les deux mains cachées. Chaque carte posée par la suite réduit encore l'incomplétude. En ce qui concerne le score, le but du déclarant est de réaliser au moins autant de plis que nécessite le contrat alors que le but de la défense va être de faire chuter le contrat en faisant le nombre de plis nécessaires pour cela. On pourrait croire que l'objectif pour les deux camps est de faire un maximum de levées mais ce n'est pas le cas et c'est une des caractéristiques qui rend ce jeu si complexe. Par exemple, un bon déclarant se contentera de faire 10 plis en toute sécurité si la manoeuvre pour en faire 11 risque de mettre en péril le contrat. De même le flanc se contentera d'une levée de chute s'il est probable de donner le contrat en tentant de faire chuter le déclarant de 2 ou plus. Il s'agit pour un expert de calculer des espérances de gain s'appuyant sur des probabilités (par exemple de répartition d'une couleur dans les mains cachées), mais aussi sur des facteurs psychologiques tels que la connaissance du niveau et du style de l'adversaire, la manière dont il a joué les premieres cartes etc.

Paradoxe du bridge : il n'y a aucune contrainte d'ordre ou autre sur le choix de la carte. Un joueur n'est jamais obligé de mettre une carte plus forte ou de couper. C'est en partie cette simplicité apparente qui rend le jeu de la carte si complexe.

Tâches à réaliser par un joueur humain
Un joueur humain va se trouver confronté aux problèmes suivants :

1. Le choix de l'entame : ce choix est primordial car bien souvent il conditionne la réussite ou non du contrat.

2. La mémoire : il doit se souvenir des enchères, des conventions* du jeu de la carte et des cartes jouées.

3. Le raisonnement : le joueur doit raisonner afin de déduire des informations sur les deux mains cachées.

4. La révision d'hypothèses : il doit revoir ses hypothèses par exemple lorsque celles-ci deviennent trop peu probables ou impossibles en fonction des nouvelles cartes qu'il a vues.

5. Le calcul de probabilité des lignes de jeu envisageables.

Pour résumer on peut dire que pour les deux étapes, une grande partie de la tactique de jeu consiste à reconstruire l'information manquante à partir du contexte.
Avant de rentrer dans les détails liés aux caractéristiques du bridge par rapport aux notions de théorie des jeux, nous insistons sur le fait que le bridge est avant tout un jeu de raisonnement dans lequel l'apprentissage est central et permanent. Chaque tournoi est un enseignement y compris pour des champions et il n'existe aucun championnat où un joueur n'a fait aucune faute.

3. Bridge informatisé

Nous présentons un bilan des résultats obtenus par la communauté *computer bridge* qui s'intéresse depuis une quarantaine d'années à ce sujet en commençant par décrire le bridge d'un point de vue théorie des jeux. Nous faisons un historique des programmes de bridge et donnons les principes généraux des programmes actuels. Enfin, nous décrivons quelques pistes de réflexion sur les différents challenges inhérents aux IA de bridge.

3.1. Bridge et théorie des jeux

Les jeux sont classés en catégories qui se distinguent relativement à des caractéristiques ayant une influence sur la manière de traiter le jeu. Nous donnons les caractéristiques du bridge relativement à ces notions classiques, ces propriétés sont valables aussi bien pour la phase d'enchères que celle du jeu de la carte.

Le bridge est un jeu :

– **A information incomplète** : les joueurs ne voient qu'une partie des cartes, ils ont donc une connaissance incomplète de l'état. Ils ne connaissent pas non plus les gains résultant de leurs actions. Cependant chaque joueur connaît ses possibilités d'action, les possibilités d'action des autres joueurs et leurs motivations.

– **A mémoire parfaite** : ce n'est pas le cas dans un cadre réel mais on va supposer que chaque joueur peut se rappeler à tout moment la suite de coups qui ont été joués précédemment.

– **Non coopératif** : les joueurs ne peuvent se concerter pour définir des stratégies.

– **Séquentiel** : pour les 2 phases, l'ordre des actions des joueurs est déterminé (sens des aiguilles d'une montre).

– **Fini** : l'ensemble des stratégies de chaque joueur est fini.

– **A somme constante** : pas exactement à somme zéro mais à somme constante puisque les deux camps se battent pour treize plis.

– **Répété** dans le cadre d'un ensemble de donnes. La répétition permet de modifier le comportement d'une paire face à une autre. Un joueur peut modifier ses enchères et son jeu de la carte en fonction du niveau estimé des adversaires mais aussi en fonction de son partenaire. Si le joueur constate au fil des donnes que son partenaire est optimiste, il va être plus prudent pour éviter de jouer des contrats trop élevés. On connaît toujours le nombre de parties qu'on va jouer mais suivant les cadres de jeu on peut jouer contre les mêmes paires ou non. Un joueur peut aussi modifier son comportement s'il évalue qu'il a beaucoup d'avance ou de retard dans le match.

3.2. Historique et description des programmes de bridge

Il existe deux grandes catégories de programme de bridge. Les premiers appelés *superviseurs* fournissent un ensemble d'outils permettant aux joueurs humains de jouer entre eux via internet. Les superviseurs peuvent être utilisés dans un cadre expérimental de recherche. La seconde catégorie regroupe des *logiciels de jeu* permettant de progresser en jouant seul. Dans certains cas, le joueur dispose d'un nombre limité de donnes choisies par un expert pour leur intérêt pédagogique. Ces donnes pourront être rejouées interactivement et éventuellement accompagnées de commentaires du professeur. D'autres logiciels permettent de jouer un nombre illimité de donnes avec ou contre des robots de bridge. Il n'y a pas de commentaires permettant de faire progresser l'utilisateur les donnes étant choisies aléatoirement. Le Graal pour les logiciels de jeu serait de permettre de jouer un nombre illimité de donnes avec des commentaires associés.

Dans la suite, nous nous intéressons uniquement aux robots de bridge (voir aussi (Paul, 2010) pour un état de l'art sur le bridge informatisé).

Depuis 1996, les meilleurs programmes du monde se rencontrent tous les ans dans le cadre des championnats du monde de bridge des robots (WBBC). Cette compétition est organisée par la Fédération Mondiale de Bridge (WBF, Président Gianarrigo Rona) et la Fédération américaine (ACBL), sous la direction d'Alvin Levy (https://bridge-robots.com/).

Avant l'existence de ces championnats et dès les années 1960 sont apparus des algorithmes permettant de résoudre des problèmes de bridge extrêmement limités par la faible puissance de calcul (Berlekamp, 1963 ; Wasserman, 1970). BridgeBaron, paru en 1983, fut le premier logiciel permettant de jouer des donnes entières mais à un niveau encore très faible.

Face à un problème de jeu de la carte, le joueur humain construit un plan de jeu éventuellement divisé en étapes mettant en œuvre des techniques de bridge répertoriées : impasse*, affranchissement d'une couleur*, etc. (Smith *et al.*, 1996) ont utilisé avec un certain succès des techniques de planification (Hierarchical task network)

ayant pour but de déterminer les techniques à mettre en place. Leur approche a permis à BridgeBaron de remporter les WCBC en 1997, tout en jouant à un niveau sensiblement plus faible qu'un joueur moyen de bridge amateur.

Dans le cadre des programmes actuels, la technique la plus connue pour le jeu de la carte est appelée *double dummy* ou *jeu de la carte à 4 jeux*. Elle consiste à définir un solver maximisant le nombre de plis pour chaque camp sur une version simplifiée du bridge. Un solver de ce type est la base de tout programme de bridge puisqu'il est utilisé comme fonction d'évaluation.

Dans cette version simplifiée le contrat et les 4 mains sont connus et on ignore volontairement les enchères. Nous sommes alors dans un cadre information complète ce qui rend la version *double dummy* déterministe. Un arbre de recherche peut donc être construit sur ce jeu simplifié et il devient possible d'utiliser les techniques d'élagage classiques telles que l'algorithme Alpha-beta (α-β). Les valeurs des feuilles sont obtenues en utilisant le résultat du solver à 4 jeux. Comme les plis remportés le sont définitivement, le nombre de plis courant fournit des bornes inférieure et supérieure pour l'élagage α-β avec des intervalles de plus en plus petits. Malgré cela, le facteur de branchement est encore trop grand pour permettre une résolution des problèmes de carte par la force brute. Les programmes de bridge ont commencé à utiliser une méthode comprenant deux phases : réduction de l'espace d'états en utilisant les symétries du jeu de bridge (typiquement en mettant des 'petites' cartes [5] de la même couleur dans la même classe d'équivalence) puis résolution à quatre jeux d'un échantillon de donnes générées par une méthode de Monte-Carlo en cohérence avec les enchères produites et les cartes jouées. Le facteur de branchement est substantiellement réduit, passant de b à b x 0,76. Cette méthode connue sous le nom de *partition search* a été formalisée dans (Ginsberg, 1996) et décrite de manière plus approfondie dans (Ginsberg, 2001). Le logiciel GIB (*Ginsberg's Intelligent Bridge player*) créé par Ginsberg a remporté les WCBC en 1998 et 1999.

Sur internet, le plus connu des jeux en ligne s'appelle *Bridge Base Online* (BBO représenté figure 1). La société Bridge Base Inc le développant a été eté fondée en 1990 par Frederick « Fred » Gitelman. BBO est gratuit et a la particularité de fonctionner comme un superviseur mais également de permettre aux joueurs d'utiliser des robots de type GIB.

3.3. Monte-Carlo dans le bridge

Le principe de la recherche Monte-Carlo dans le cadre des jeux est de faire des statistiques sur les actions possibles à partir de parties jouées aléatoirement. Dans les jeux à information incomplète l'approche Monte Carlo semble la meilleure puiqu'elle permet de générer aléatoirement la complétion de l'information cachée. L'arbre de recherche est l'outil incontournable de tout jeu à information complète, couplé à des

5. La notion de petite carte est floue, selon les cas elle peut concerner les cartes du 2 au 10, ou du 2 au 5.

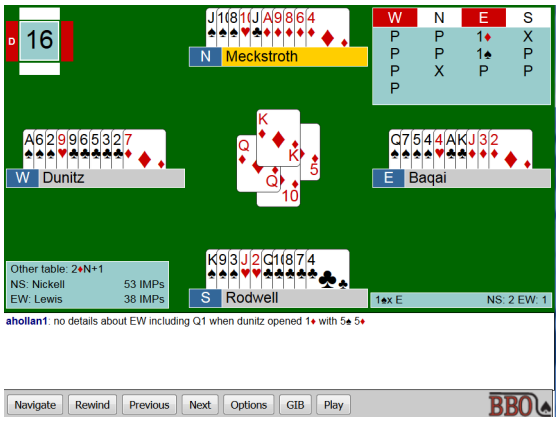


Figure 1. Retransmission BBO. En haut à droite : la séquence d'enchères, en bas à gauche : les scores en IMP de chacune des équipes

fonctions d'évaluation et à des stratégies locales astucieuses. Toutefois, pour des jeux comme le go ayant un important facteur de branchement et où la fonction d'évaluation n'est pas aisée à construire, des simulations Monte-Carlo permettent de générer des échantillons réduits afin de prendre une décision d'action. Ces simulations sont également utiles au bridge dans deux situations : les décisions finales aux enchères et le jeu de la carte.

Enchères : les premières enchères sont souvent déterminées par un système de règles couplé à un recensement de situations classiques. En revanche, une telle architecture ne permet pas de couvrir toutes les séquences d'enchères (de l'ordre de 10^{47} séquences d'enchères possibles au bridge) et, quand on approche de la décision du contrat final, une simulation est utile afin de générer des mains (celle du partenaire et celles des adversaires) cohérentes avec les enchères précédentes ; l'enchère gagnante dans le plus grand nombre de cas est ainsi sélectionnée.

Jeu de la carte : après l'entame le déclarant voit sa main et celle de son partenaire ; il doit imaginer, en fonction des enchères précédentes et de l'entame les jeux de ses adversaires et prendre des décisions successives afin de mener à bien son contrat. Les logiciels de bridge simulent des échantillons de mains pour les jeux adverses et sélectionnent l'action gagnante dans le plus grand nombre de cas. Cette approche a des résultats généralement satisfaisants mais présente de nombreuses lacunes : si la simulation prend en compte les enchères (ou absence d'enchères) des adversaires, elle ne prend pas (encore) en compte les actions initiales des adversaires au jeu de la carte. Là où un joueur humain raisonnera que si son adversaire joue de telle façon il a tel ou tel jeu, les logiciels de bridge sont actuellement incapables de mener ce genre d'analyse.

3.4. Fonctionnement des IA de bridge actuelle

La plupart des IA de bridge disposent des modules suivants :

– M1 : Générateurs de donnes sous contraintes liées aux caractéristiques des mains. De nombreux programmes de type M1 ont été réalisés dans les années 1990.

– M2 : Algorithme efficace de résolution à 4 jeux qui permet d'avoir un calcul automatique du *Par d'une donne** à partir des 4 mains.

– M3 : Module de production des enchères.

– M4 : Module inverse permettant de traduire les enchères en contraintes.

– M5 : Evaluation de l'espérance de gain en IMP.

– M6 : Choix de l'entame.

Le module M2 est un solveur à quatre jeux qui est utilisé comme fonction d'évaluation aussi bien pour le jeu de la cartes que pendant les enchères. Généralement, les concepteurs de programmes de bridge définissent leur propre solveur, mais on peut utiliser celui développé par Bo Haglund connu comme le plus rapide et le plus utilisé (http://privat.bahnhof.se/wb758135/bridge/index.html). En combinant M1 et M2 on obtient un module de test qui permet de donner une approximation sur le pourcentage de réussite d'une ligne de jeu en supposant que tout le monde joue à cartes ouvertes et sur le calcul de l'espérance de gain approximative d'un contrat donné.

Pour le module M3, les programmes disposent d'un ensemble de règles qui permettent de choisir une enchère à partir du contexte. Par exemple, on choisira l'enchère de 1SA pour les mains sans singleton* de 15 à 17 points d'honneur *. La simulation intervient le plus tard possible idéalement pour déterminer le contrat final. Elle peut aussi intervenir lorsqu'on rencontre des situations non prévues (séquence d'enchères non répertoriée).

Le module M4 est utilisé pour ne pas générer de main incompatible avec les enchères précédentes. Le module M5 utilise les sorties des modules précédents pour évaluer les différentes options. Enfin, le module M6 concerne *l'entame* qui est un compartiment du jeu à part où les programmes utilisent des stratégies différentes. En voici quelques-unes :

– choisir la carte naturelle pour chaque couleur et faire ensuite des simulations sur ces 4 choix.

– choisir une règle parmi un ensemble de règles en tenant compte du contexte. Si le choix n'est pas unique sélectionner la meilleure entame en faisant des simulations.

– préférer la couleur la plus longue.

Certains choix de concepteurs sont liés à d'autres critères que l'optimalité immédiate. En effet, ils peuvent écarter une entame un peu meilleure statistiquement mais très étrange au profit d'une autre plus classique qui sera plus appréciée par le joueur humain partenaire ou adversaire du robot. Les concepteurs ne dévoilent pas facilement leur code et il n'y a pas beaucoup de documentations sur le fonctionnement des programmes. La communauté constate depuis quelques années que les progrés des robots

sont très marginaux. Ceux-ci progressent toujours, mais en couvrant des situations particulières qu'ils géraient mal auparavant et qui sont codées en dur dans le programme.

4. Challenges pour les programmes de bridge

La principale tâche consiste à compléter l'information manquante à partir du contexte. Le contexte varie suivant les étapes. Pour la partie enchères, il s'agit de la séquence d'enchères courante. Pour la partie jeu de la carte le contexte comprend la séquence d'enchères mais aussi les cartes déjà jouées. Sur la partie jeu de la carte, on peut assez facilement évaluer chaque position et le résultat final. Pour la partie enchères, l'évaluation d'une séquence d'enchères en cours n'a pas de sens. Par exemple, il existe des enchères artificielles qui ne traduisent pas une intention de jouer le contrat relatif à l'enchère. On peut par contre évaluer le résultat c'est à dire évaluer le contrat final auquel on aboutit.

Challenges enchères

1. Apprendre automatiquement une nouvelle manière de compter les points permettant de prendre des décisions avec un système de règles associé.

2. Apprendre à réagir en face d'un partenaire donné et mettre à jour automatiquement la base de règles en fonction du partenaire. Ce challenge est intéressant pour les robots jouant en face d'humains car ceux-ci se lassent vite de l'incompréhesion de leur partenaire robot.

3. Revoir la fonction d'évaluation liée à la méthode à 4 jeux qui n'est pas satisfaisante. En effet, celle-ci va compter comme gagnant un contrat s'il existe au moins une ligne de jeu gagnante avec la meilleure défense possible. Une donne sur laquelle le contrat ne gagne qu'en utilisant une ligne de jeu très peu probable que l'on peut qualifier de mal jouée sera comptée comme positive alors qu'il est certain que tous les experts ne réussiront pas leur contrat puisqu'ils choisiront une ligne perdante mais plus prometteuse d'après les probabilités. On peut envisager d'associer des poids aux lignes de jeu en fonction de ces critères probabilistes pour pondérer le résultat de la fonction d'évaluation mais les calculs sont difficiles et coûteux.

4. Revoir le jugement de la main en fonction des enchères : il faudrait envisager une abstraction des mains de plus en plus fine au fur et mesure que la séquence se précise. Dans le cadre d'une approche symbolique utilisant un langage compréhensible par l'humain, on pourra utiliser des termes du langage de plus en plus précis. Par exemple, au début on peut considèrer uniquement les valeurs et couleurs des 13 cartes puis on effectue des saturations relatives par exemple à la distribution, au nombre de cartes dans la couleur principale du partenaire etc. Les opérations sont plus coûteuses sur un langage plus riche mais elles se feraient lorsque l'incertitude est plus faible.

5. Apprendre automatiquement un système d'enchères : définir un système d'enchères optimal est un problème ouvert sur lequel se penchent des théoriciens du bridge depuis de nombreuses années. Des travaux utilisant des résaux neuronaux à partir des règles du jeu de bridge n'ont rien donné de concluant.

6. Mettre des degrés d'agressivité différents en fonction des situations (partenaire, adversaire, retard dans un match). On peut affiner en mettant des degrés différents pour un même joueur sur un même match (ex : agressif en intervention, mais pas en réponse à l'intervention etc).

Challenges jeu de la carte

1. Evaluation d'une ligne de jeu* : on peut dire d'un bridgeur qu'il a très bien joué une donne* alors qu'il a fait deux plis de moins que tous les autres si le raisonnement global qu'il dit avoir suivi semble meilleur que celui des autres joueurs. L'évaluation d'une ligne de jeu* est basée sur des calculs de probabilités assez simples. Par exemple, on dit qu'une ligne de jeu est à 50 % si le succès de la stratégie mise en place par le joueur dépend du placement d'une carte dans un des deux jeux non visibles. Malgré l'apparente simplicité des calculs, il est beaucoup plus complexe d'évaluer une ligne de jeu car il faut utiliser toutes les informations dont on dispose telles que les informations liées aux enchères couplées au compte des points et au compte de distribution. Par exemple si la carte en question est dans une couleur annoncée comme courte chez un adversaire elle a moins de chances de s'y trouver.

2. Utiliser les informations liées à l'entame : au cours du jeu, les joueurs peuvent fournir des cartes trompeuses. C'est la partie « bluff » du bridge, la différence avec le poker est qu'on a un partenaire qui pourra être trompé aussi et mal raisonner ensuite. La carte d'entame est souvent considérée comme sûre.

3. Utiliser les informations liées à la manière de jouer des adversaires : pour cela il faut connaître le niveau des joueurs, on peut envisager aussi d'apprendre à évaluer le niveau ou le style d'un joueur.

4. Améliorer les techniques de flanc des robots : il y a beaucoup à faire sur cette partie qui nécessite une collaboration entre les défenseurs. La complexité provient également des conventions utilisées pour les cartes jouées.

5. Entame : comme nous l'avons vu précédemment l'entame est souvent cruciale. Il est envisageable de se concentrer uniquement sur cette partie du jeu en donnant en entrée d'un réseau profond des données (la séquence d'enchères, la main de l'entameur, l'entame). Le but du réseau sera de prédire la carte entamée parmi les treize.

6. Simulation : les programmes utilisent une approche à 2 jeux donc plus fine que le double dummy quand il reste moins de cartes. Certains programmes décident de le faire toujours au niveau du même pli (par exemple le quatrième), d'autres distinguent les cas suivant la hauteur du contrat (ex : quatrième pour une manche, troisième pour un chelem). On peut automatiser le procédé en repérant automatiquement les endroits où une analyse plus fine est nécessaire.

Challenge commun aux deux phases : reconnaissance de patterns

Il existe 5.36×10^{28} donnes différentes, et donc peu de chances de jouer une même donne. Cependant, on retrouve des situations similaires et la notion de patterns est très importante chez un bon joueur. L'expérience permet de repérer ces patterns pour les différentes phases du jeu. Des ouvrages de bridge proposent de résoudre des pro-

blèmes sur des donnes qui nécessitent la mise en place d'une certaine technique de jeu. Le lecteur pourra ensuite gérer la donne mais aussi tout un ensemble de donnes du même type. Le but est de reconnaître le pattern pour appliquer la technique associée. Pour un déclarant, il existe un ensemble de techniques connues pour maximiser le nombre de plis qu'il peut faire comme : la double coupe, le squeeze, l'affranchissement d'un couleur. Ici encore il faut pouvoir reconnaître les donnes pour lesquelles ces techniques s'appliquent soit pour les appliquer lorsque l'on est déclarant soit pour empêcher le déclarant de les mettre en place quand on est en flanc. La reconnaissance est plus difficile pour le flanc qui ne voit qu'un des deux mains du camp en attaque. Un bon défenseur se sert des enchères pour imaginer la main cachée et deviner quelles techniques le déclarant va essayer de mettre en place. Il se sert également des informations liées aux cartes déjà jouées.
Un challenge important pour le bridge est de pouvoir retrouver ces patterns pour pouvoir mettre en place des stratégies prévues par le système ou apprises.

Dans la suite de l'article, nous présentons nos travaux ayant permis de booster une IA existante.

5. Boosting d'une IA de bridge

Les approches proposées dans (St-Pierre, Teytaud, 2014 ; Cazenave *et al*., 2015 ; Liu *et al*., 2016) ont montré que le choix de la graine aléatoire pouvait avoir un impact non négligeable sur les performances d'une l'IA utilisant des simulations de Monte-Carlo. Pour cela, les auteurs ont défini et testé des algorithmes permettant de choisir de bonnes graines ou de générer de bonnes distributions de probabilité sur les graines. Dans certains cas l'optimisation est quasi nulle, dans d'autres elle est très significative. Le but de nos travaux est d'adapter la méthode de boosting à une IA de bridge et d'étudier l'impact du choix de la graine aléatoire sur la performance de l'IA.

5.1. Approche de Boosting IA de jeux

Un programme stochastique utilise une graine aléatoire qui détermine la séquence de nombres générés à chaque appel de la fonction random dans la suite du programme (on parle de séquence pseudo-aléatoire). Cette graine peut être générée aléatoirement par exemple en utilisant l'horloge ou fixée de manière arbitraire par le concepteur.

Le principe général de l'approche présentée dans (St-Pierre, Teytaud, 2014) est de construire des IA déterministes à partir d'une IA stochastique et de comparer les variations des résultats en faisant jouer les IA déterministes les unes contre les autres.

L'idée est qu'un programme stochastique est en fait une variable aléatoire distribuée sur un programme déterministe. Soit IA(w) le programme stochastique, IA(1) correspond au programme déterministe de la meme IA utilisant la graine 1.

Deux algorithmes de simulation pour des jeux à deux joueurs ont été définis : *BestSeed* qui permet de trouver la meilleure graine et *Nash* qui renvoie une distribution

de probabilité sur les graines. Une extension de la méthode est donnée dans (Liu *et al.*, 2016) et testée sur trois nouveaux jeux Atari-Go, Domineering un jeu combinatoire à base de dominos de différentes tailles, et Breakthrough une variante des échecs.

Le bridge n'est pas dans ce cadre d'information complète mais l'approche a été récemment testée avec succès sur un jeu partiellement observable. En effet, dans (Cazenave *et al.*, 2016) et (Pepels *et al.*, 2015) les auteurs ont réalisé une expérimentation sur une IA de Phantom Go [6]. Les gains sont les suivants : sur un plateau de taille 5X5 le taux de victoires est passé de 50 % à 70 % contre la même IA non boostée et de 0 % à 40 % contre une IA plus forte sur Phantom Go (5X5, 7X7 et 9X9).

5.2. Adaptation au bridge

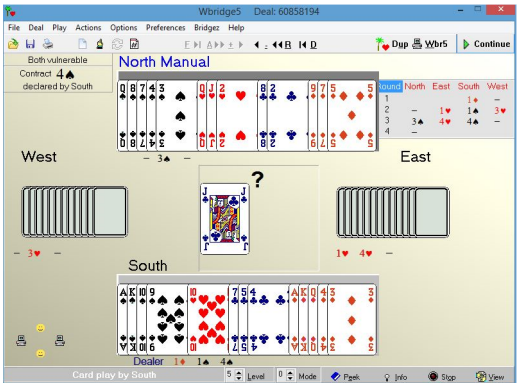


Figure 2. Copie d'écran de Wbridge5

Le but est de savoir si la méthode d'optimisation de la graine peut être efficace sur les IA de bridge et si oui sur quelles parties du raisonnement. L'IA que nous avons sélectionnée est celle développée par Yves Costel pour Wbridge5 (figure 2). Wbridge5 répond aux conditions pour que la méthode puisse être appliquée. Le programme est stochastique et utilise un générateur de nombres pseudo-aléatoires pour obtenir un un échantillon des mains cachées. Le programme permet de fixer la graine du générateur ce qui rend le programme déterministe. Nous pouvons donc construire une matrice en faisant jouer les IA déterministes les unes contre les autres.

La première difficulté est de gérer le fait qu'on l'on ait quatre IA et non pas deux comme dans les jeux traités jusqu'ici. Nous devons donc fixer quatre graines. Pour rendre plus simple l'interprétation des résultats, nous avons choisi de fixer une graine pour chaque camp et non pas une pour chaque joueur. Par ailleurs, il n'y a pas de raison a priori pour que la graine soit la meilleure pour les deux étapes de jeu pour

6. Phantom Go est un jeu à deux joueurs à information incomplète qui consiste à jouer au go sans voir les mouvements de l'adversaire.

lesquelles les simulations ne sont pas utilisées de la même manière. Nous avons choisi de commencer par une optimisation de la graine utilisée dans la phase des enchères, le contrat final étant crucial car même si le jeu de la carte est parfait, on obtiendra un mauvais score si le contrat joué n'est pas le bon. Il y a plus de possibilités dans la partie enchères car 3 mains soit 39 cartes sont inconnues tandis qu'il y a seulement 25 cartes inconnues au maximum pendant le jeu de la carte après l'entame. En outre, le jeu de la carte bénéficie des renseignements déduits des enchères et des cartes déjà jouées. Il y a donc beaucoup moins d'inconnues. Les derniers plis sont pratiquement joués à cartes ouvertes. Le temps de calcul consacré au jeu de la carte est également beaucoup plus long que celui consacré aux enchères.

5.3. Les scores au bridge

Les résultats possibles des jeux traités dans (Cazenave *et al.*, 2015 ; Liu *et al.*, 2016) comme Domineering, Atari-Go, ou Phantom-Go sont 0 pour une défaite, 0.5 en cas d'égalité et 1 pour une victoire. Le bridge necessite des résultats plus précis. En effet, il est important de connaître la différence exacte de points entre le vainqueur et le perdant. Le score d'une donne est relatif au contrat atteint à l'issue des enchères et au nombre de plis gagnés pendant le jeu de la carte.

Figure 3. Boîte à enchères utilisée pendant les compétitions de bridge

Scorage d'une donne

Soit n un nombre entier compris entre 1 et 7 et S $\in$ (♠, ♡, ♢, ♣, SA). Le contrat nS [7] est un contrat où le nombre minimum de plis à réaliser par le déclarant est égal à (n + 6) avec S représentant la couleur d'atout s'il y en a une ou le fait qu'il n'y ait pas d'atout si S = SA (Sans Atout). Par exemple, 4♡ sera gagné par le camp en attaque si le déclarant a remporté au moins 10 plis en jouant à l'atout ♡. Si le déclarant réalise 10 plis le camp en attaque obtient +420 (ou +620 suivant la marque), si le déclarant a remporté moins de 10 plis l'autre camp marque +50 (ou + 100) par levée manquante.

Nous nous focalisons sur la marque duplicate qui est utilisée pour réduire de manière significative le facteur chance. Le principe est le suivant : dans un match s'af-

7. Les différents contrats possibles sont représentées sur la figure 3.

frontent deux équipes A et B composées de deux paires chacune. Les mêmes donnes sont jouées à deux tables différentes, les paires de la même équipe étant installées dans des directions opposées. Les joueurs sont représentés par les points cardinaux Nord, Est, Sud et Ouest (abrégés en N, E, S, O) les paires pouvant être soit NS soit EO. Pour chaque donne du match, la paire NS (resp. EO) de l'équipe A rencontrera la paire EO (resp. NS) de l'équipe A et aura exactement les mêmes mains que la paire NS (resp. EO) de l'équipe B à l'autre table.

Le score final pour une donne consiste à calculer la différence des scores aux deux tables. Par exemple, si l'équipe A a marqué + 420 et l'équipe B - 50, la différence est de 470 en faveur de A. Cette différence est ensuite convertie en *International Match Point scale* (IMP). La conversion non linéaire des points en IMP est représentée figure 4. Une équipe peut obtenir entre -24 + 24 IMP par donne. Le score d'un match est la somme des scores en IMP de chaque donne.

Diff. in Pts.	IMPs	Diff. in Pts.	IMPs	Diff. in Pts.	IMPs	Diff. in Pts.	IMPs
20 - 40	1	270 - 310	7	750 - 890	13	2000 - 2240	19
50 - 80	2	320 - 360	8	900 - 1090	14	2250 - 2490	20
90 - 120	3	370 - 420	9	1100 - 1290	15	2500 - 2990	21
130 - 160	4	430 - 490	10	1300 - 1490	16	3000 - 3490	22
170 - 210	5	500 - 590	11	1500 - 1740	17	3500 - 3990	23
220 - 260	6	600 - 740	12	1750 - 1990	18	4000 and up	24

Figure 4. Table de conversion des points en IMP

Pour plus de détails sur la marque :

`https://fr.wikipedia.org/wiki/Marque_du_bridge`

5.4. Expérimentations : choix de la meilleure graine

Les expériences ont été réalisées par Yves Costel en utilisant le moteur de bridge de Wbridge5 sur un Dell Ultra book INtel 2.4 Ghz. Nous nous plaçons dans le cadre d'un match par 4 qui oppose deux équipes de 4 joueurs. Le match par quatre est la compétition reine du bridge et c'est avec ce format que les championnats des programmes sont joués. Nous utilisons la marque IMP décrite dans la section précédente.

Avant de construire une matrice complète nous avons étudié l'éventuel impact de la graine sur un match de 64 donnes (c'est le nombre utilisé pour les matches des World Computer-Bridge Championship). Le résultat de chaque donne est calculé à partir du Par de la donne c'est à dire à partir du résultat théorique du jeu à cartes ouvertes et non pas en faisant jouer les 8 robots. L'expérience concerne donc un choix de graine pour les enchères, la graine choisie ne participant pas au processus du jeu de la carte. Nous avons constaté sur ces donnes que la graine n'avait une influence que dans environ 10 % des cas mais que le gain pouvait être important car ces cas concernaient des

décisions tendues (du type « faut-il demander la manche ou le chelem ? ») qui peuvent rapporter ou coûter une dizaine d'IMP sur une seule donne.

Ce résultat étant concluant, nous avons entrepris de construire une matrice de taille 40 x 40 où chaque cellule R[i,j] contiendra le résultat en IMP d' un match de 64 donnes entre deux instances du programme déterminisées avec la graine i et deux instances déterminisées avec la graine j.

REMARQUES.
1. Durant les expérimentations, nous avons observé qu'il n'y avait aucune simulation pour 10 % des donnes, et que dans 75 % des cas le contrat atteint était le même pour les deux graines.
2. La graine n'a un impact que dans 15 % des cas mais ces cas concernent des prises de décisions importantes avec beaucoup d'IMP en jeu (ces donnes sont appelées donnes à swing).
3. Moins de la moitié des termes est calculée puisque R[j,i]=-R[i,j] et la diagonale est constituée de zéro car elle représente les matches opposant les 8 mêmes IA.

Le vecteur suivant représente les scores cumulés (somme de toutes les valeurs par colonne) par graine testée :
(156 -28 55 167 -170 -188 -16 246 -30 94 -111 -113 266 -73 13 110 54 63 86 21 -53 -263 30 130 -62 -162 -57 -20 -173 -5 -81 57 92 -151 -25 -44 118 -234 269 32).
Plus le résultat est grand, meilleure est la graine. Sur ces expériences, la plus mauvaise graine est la graine 22 (- 263) et la meilleure la graine 39 (+269).

Nash renvoie une distribution de probabilité sur les graines ce qui permet de faire en sorte que l'IA ne soit pas prévisible pour l'adversaire. Ce n'est pas le problème pour le bridge en raison de sa complexité. Nous avons donc utilisé et testé l'approche BestSeed qui est directement adaptable à des jeux de plus de 2 joueurs et nous avons sélectionné la meilleure graine relativement à la somme de toutes les valeurs par ligne de la matrice. La meilleure graine (39 sur notre expérience) est celle dont la somme sur la ligne est la plus grande.

5.5. Validation croisée

Le protocole suivi pour la validation est celui préconisé dans : https://www.lri.fr/ teytaud/loo.pdf. L'idée générale de cette validation croisée est d'étudier l'impact de la meilleure graine en gardant une dizaine de points sur les quarante pour former l'ensemble test (pour K=1, 2, 3,..., 30) dans l'algorithme Evaluation.

Algorithme Evaluation

```
Pseudo-code :
nbArms = param
M= matrice des  r\'esultats
pour K=1, 2, 3,..., 30
{
   pour i=1, 2,..., n        (n = minimum 500)
    {
       M=M(randompermutation(40), randompermutation(40))
       submatrix=M( 1:K,   1:K )
       testmatrix=M( 1:K, (K+1):40 )
       score(1) = somme de la  ligne 1 de M
       score(2) = somme de la  ligne 2 de M
     ...
        score(K) = somme de la ligne K  de M
        soit I=(i1,...,inbArms)
        la liste des  indices avec le meilleur score
        resultats(i) = moyenne des lignes dans testmatrix
    }
      performance(K)=moyenne des resultats(i)
}
```

Les courbes ci-dessous représentent le résultat de l'implémentation en Python de l'algorithme Evaluation. Dans un premier temps, nous avons étudié la performance de la meilleure graine sur une nouvelle matrice où toutes les valeurs positives de la matrice initiale ont été remplacées par la valeur 1 représentant une victoire et toutes les valeurs négatives de la matrice par la valeur -1 représentant une défaite.

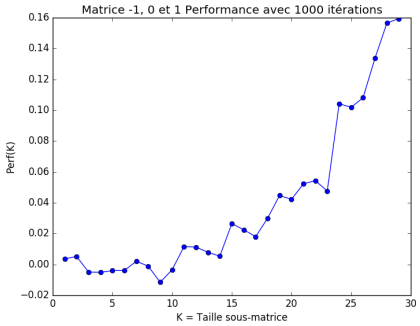


Figure 5. Courbe de performance cadre -1,0 et 1 avec 1000 itérations

La figure 5 représente l'évolution de la performance de la meilleure graine dans le cadre -1, 0 et 1. La performance augmente de manière régulière et significative.

La progression de la performance est moins stable dans le cadre IMP (figure 6) testé avec 3000 itérations. ce qui peut s'expliquer par le fait que la graine a une influence qui provoque de gros écarts mais sur peu de cas comme on l'a vu précédemment.

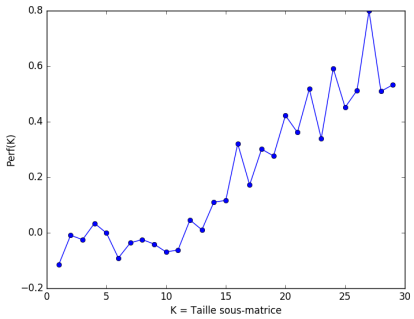


Figure 6. Courbe de performance cadre IMP avec 3000 itérations

Nous avons vérifié que l'écart type diminuait dans le cadre IMP, ce qui est le cas (cf figure 7).

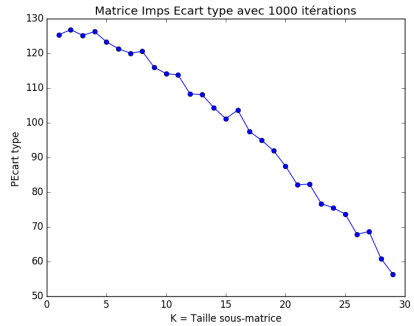


Figure 7. Ecart type cadre IMP avec 1000 itérations

5.6. Validation interne

Un match a été lancé dans les mêmes conditions mais cette fois sur 1000 donnes entre le programme avec la meilleure graine trouvée (39) et le programme avec la plus mauvaise (22). Le programme avec la meilleure graine a gagné avec un écart de 163± 2,62 IMP, sur 1000 donnes, i.e. 0,16 imp par donne. Un second match de 1000 donnes entre le programme avec la meilleure graine (39) et le programme avec la graine actuelle (99) qui avait été choisie arbitrairement a donné les résultats suivants. Le programme avec la meilleur graine a remporté le match avec un écart de 97 ±

2,75 IMP sur 1000 donnes, ce qui correspond à un gain de $\simeq$0,1 imp par donne. Ces résultats confirment la validation croisée. Dans la matrice chaque graine est testée sur 2496 donnes (64*39), la meilleure graine gagnant 269 IMP et la plus mauvaise perdant 263 IMP soit légèrement plus de 0,1 imp par donne.

A l'issue de ces expérimentations, nous avons décidé de remplacer la graine courante par la meilleure graine pour la participation de Wbridge5 aux championnats du monde de computer bridge.

5.7. Championnats du monde septembre 2016

La vingtième édition des WCBC [8] a eu lieu à Wroclaw (Pologne) du 5 au 10 septembre 2016 sous la direction d'Alvin Levy qui fournit les donnes, arbitre en cas de litige, et établit le classement. La partie technique est entièrement gérée par Gérard Joyez qui a développé *Table Manager* le programme central de ces championnats puisqu'il permet de gérer l'échange d'informations entre les robots et d'éviter tout système de triche. Voici les huit participants avec leur pays d'origine et leur palmarès : Bridge Baron (USA, 1997), Meadowlark Bridge (USA, 2000), Micro Bridge (Japon), Q-Plus Bridge (Allemagne), RoboBridge (Pays-Bas), Shark Bridge (Danemark, 2011 et 2014), Wbridge5 (France, 2005, 2007 et 2008) et Xinrui (Chine). On notera l'absence du champion du monde en titre Jack (Pays-Bas) et le retour de Meadowlark Bridge après des années d'absence ainsi que l'arrivée d'un nouveau compétiteur Yuzhang Liu dont le robot Xinrui est issu de d'une entreprise chinoise de 20 personnes dédiée au computer bridge et dont il est le fondateur.

Le format de la compétition consiste en une phase qualificative appelée Round-Robin au cours de laquelle tous les robots s'affrontent dans des matchs de 32 donnes. A l'issue de cette phase, les quatre premiers programmes sont qualifiés pour les demi-finales le premier rencontrant le quatrième et le second le troisième dans des matchs de 64 donnes. Les vainqueurs de ces demi-finales jouent ensuite la finale sur un match de 64 donnes. Résultats du Round Robin :
1. Wbridge5 91,87
2. Micro Bridge 90,07
3. Bridge Baron 89,21
4. Shark Bridge 80,12
5. Q-Plus Bridge 78,76
6. Xinrui 78,50
7. RoboBridge 50,58
8. Meadowlark Bridge 0,89

Wbridge5 a terminé en tête du Round-Robin avec un écart très faible sur le deuxième et le troisième. Meadowlark aprés 12 ans d'absence a connu beaucoup de problèmes

8. www.computerbridge.com pour plus d'informations.

tant techniques que bridgesques. En revanche, Xinrui le nouvel arrivant a bien figuré. Il était même en passe de se qualifier avant de subir une lourde défaite face à Shark Bridge dans son dernier match. En règle générale la compétition a été serrée entre les six premiers. Ainsi Q-plus bridge n'a perdu la quatrième place qualificative que lors de la dernière donne de son dernier match.

Les demi-finales et la finale ont été également très disputées. Wbridge5 a gagné sa demi-finale contre Shark 140,6-131. L'écart de 9,6 IMP correspond en partie au carry-over gagné grâce la victoire de Wbridge5 contre Shark pendant le Round-Robin. L'autre demi-finale a vu Micro-Bridge battre Bridge Baron par un écart encore plus faible : 144-138 alors qu'à la mi-temps du match, Bridge-Baron était en tête de 66 IMP (96 à 30).

La finale a été également très serrée. A deux donnes de la fin, Wbridge5 avait 17 IMP de retard sur Micro Bridge. Cependant, les deux dernières donnes lui ont permis de gagner 23 IMP pour un score final de 162 à 156. Sur ces deux donnes l'IA a fait la différence en trouvant un meilleur contrat que son adversaire. Comme on l'a vu dans le cadre de nos expériences il semble que la graine a un impact sur des décisions tendues comme celles-ci. La différence de 6 points pour 64 donnes correspond à un gain de 0,09375 par donne alors que le gain estimé de la meilleure graine par rapport à l'ancienne est de 0,1 IMP par donne.

Wbridge5 a ainsi gagné son quatrième titre de champion du monde huit années après sa dernière victoire.

6. Perspectives et travaux en cours

La génération d'enchères a été étudiée précédemment dans (Amit, Markovitch, 2006 ; DeLooze, Downey, 2007) avec l'élaboration d'un algorithme PIDM (Partial Information Decision Making) utilisé pour prédire des enchères qualifiées de raisonnables. Dans (DeLooze, Downey, 2007), un réseau de neurones auto-adaptatif a été utilisé pour enchérir des mains restreintes aux contrats sans atout. Une grande partie des recherches actuelles sur les jeux utilise les réseaux neuronaux profonds (par exemple, (Moravčík *et al.*, 2017) pour le poker), avec des succès nets comme celui de Libratus, intelligence artificielle développée par l'Université Carnegie Mellon, qui a battu quatre des meilleurs joueurs de poker Texas Hold'em sans limite sur 20 jours de compétition. Dans le cadre du bridge les réseaux profonds ont été utilisés dans (Yegnanarayana *et al.*, 1996) et plus récemment dans (Ho, Lin, 2015 ; Yeh, Lin, 2016). Dans (Yeh, Lin, 2016), un modèle d'apprentissage par renforcement profond a été conçu pour faire de la génération automatique d'enchères dans un cadre restreint aux séquences non compétitives. Cette restriction impose que les adversaires ne participent pas à la phase d'enchères. Notre approche n'est pas en concurrence avec ces méthodes, ce type de séquences ne se produisant que dans 27 % des cartes (statistiques obtenues à partir de données sur le site de la WBF). En outre, le sous-problème traité est lié à un processus à deux joueurs plutôt qu'à quatre joueurs.

Perspectives

L'expérience a été réalisée par Yves Costel et s'est déroulée sur un Dell Ultra book Inspiron Intel 2,4 Gz. On peut envisager de l'étendre en utilisant des machines plus puissantes. Dans l'approche que nous avons mise en place la graine est la même pour toutes les donnes et tous les appels de simulation. Pendant la phase d'enchères, les situations sont répertoriées et classées suivant le nombre de plis que l'on pense faire (zone de partielle, manche ou chelem). Les IA de bridge sont actuellement capables de savoir dans quelle situation elles se situent au moment des prises de décision. On pourrait améliorer l'efficacité de l'approche en recherchant une meilleure graine pour chaque situation spécifique. On pourrait aussi distinguer les situations en fonction du nombre de possibilités. Par exemple, lorsqu'il y a eu beaucoup d'enchères les mondes possibles sont moins nombreux puisque chaque enchère crée une contrainte supplémentaire. Il semble intéressant de construire des matrices avec quatre graines différentes et bien entendu d'étendre la recherche de la meilleure graine sur le jeu de la carte qui utilise aussi des simulations.

Avant de considérer d'autres méthodes pour le bridge il est important de déterminer les sous-problèmes du bridge pour lesquels une approche de type AlphaGo pourrait être efficace. Par exemple, il est intéressant de voir si les techniques récentes de gestion et d'analyse de gros volumes de données (*big data*) utilisées par AlphaGo peuvent être appliquées au bridge. Pour cela il faut pouvoir disposer d'un grand nombre de parties de qualité jouées par des humains. C'est le cas pour le bridge car il existe de nombreuses archives de matches joués dans le cadre de championnats nationaux ou internationaux de haut niveau. A la différence du Go par exemple où une partie ne peut être jouée plusieurs fois, ces données permettent de comparer ce qu'ont fait des joueurs différents sur exactement les mêmes donnes. Les formats de ces archives sont normalisés et facilement utilisables, la difficulté principale est d'y avoir accès et surtout de posséder l'expertise suffisante pour pouvoir les trier (par exemple suivant le système d'enchères* utilisé ou suivant la catégorie lié au niveau du contrat dans laquelle on peut les mettre), et pour pouvoir définir les bonnes caractéristiques pour modéliser les donnes. La première étape d'AlphaGo permettant au système d'atteindre un bon niveau en utilisant des parties jouées par des champions nous semble prometteuse pour le bridge car l'apprentissage humain utilise beaucoup l'imitation. D'autre part, il est très important pour la progression d'un joueur qu'il puisse s'entraîner avec et contre des joueurs ayant un niveau supérieur au sien. Cependant même en admettant que l'on puisse développer une IA de bridge exactement sur le modèle d'AlphaGo, elle ne sera pas complètement satisfaisante car au bridge, il faut non seulement effectuer les bonnes actions mais il faut également pouvoir expliquer pourquoi on les a faites. L'évaluation de la stratégie d'un joueur est indépendante du résultat obtenu. On peut dire qu'un joueur a très bien joué une donne tout en sachant que le score qu'il a obtenu sur cette donne est mauvais. AlphaGo n'utilise pas la force brute classique, mais il ne peut pas expliquer le raisonnement qu'il a suivi pour choisir telle ou telle action, l'approche adoptée reste quantitative et non qualitative. Le fait de coupler des approches numériques et symboliques pourrait permettre d'avoir une approche quali-

tative du bridge. On pourrait par exemple entraîner le robot à prendre des décisions en utilisant des approches réseaux profonds et RL et utiliser l'apprentissage supervisé symbolique pour mettre à jour automatiquement les bases de connaissances des robots. Ceci permettrait la transmission du mode de raisonnement et la caractérisation de l'adaptation du robot à un partenaire différent.

Travaux en cours

La communauté computer bridge n'a pas bénéficié jusqu'ici de grandes ressources alors qu' il faudrait disposer d'une grosse puissance de calcul pour garantir que l'évaluation Monte-Carlo donne des résultats suffisamment précis. On pourrait utiliser comme pour Mogo le parallélisme à grande échelle (avec notamment GRID'5000). Nous avons commencé à étendre nos expérimentations en utilisant une méthodologie de type bandits qui nous permettra de comparer plus de graines en ne construisant qu'une partie de la matrice.

Nous travaillons également sur la définition d'un MDP (*Markov Decision Process*) et d'une adaptation d'un MCTS (*Monte Carlo Tree Search*) pour le bridge. Celui-ci servira d'entrée pour une approche de type apprentissage par renforcement. Plus précisément, nous pensons que le bridge est une bonne application pour utiliser un cadre mixte d'apprentissage par renforcement dans un cadre symbolique comme celui défini dans (Nitti *et al.*, 2015).

Les prises de décisions au début d'une donne ne peuvent être traitées par une approche Monte Carlo car les possibilités sont trop nombreuses. Dans ce cadre, on peut envisager de faire un pré-traitement qui permettrait de simuler les decisions d'un expert ou de mettre à jour la base de règles d'un robot. Des travaux relatifs à ce pré-traitement sont en cours de réalisation en collaboration avec Swann Legras. Ces travaux s'inscrivent dans un cadre d'apprentissage supervisé et sont décrits brièvement ci-dessous.

Dans un cadre d'apprentissage supervisé, la base d'exemples utilisée en entrée est partitionnée en deux ensembles [9] : E+ (les exemples positifs) et E- (les exemples négatifs). A partir de cette base, le but est soit dans un cadre numérique de simuler les décisions d'un expert en apprenant un estimateur qui renverra + ou - en imitant le comportement lié à la base, soit d'apprendre un ensemble de règles explicites permettant d'imiter l'expert tout en expliquant pourquoi on a choisi + ou -. Dans ce cadre symbolique, il sera possible de mettre à jour la base de règles d'un robot pour l'améliorer si on considère que c'est une bonne règle (l'expert est un champion incontesté), ou encore pour personnaliser la base relativement à un partenaire qui n'aurait pas les mêmes règles que l'IA ce qui peut être une avancée pour les robots jouant en face de différents partenaires humains. On peut aussi envisager d'apprendre plusieurs estimateurs (ensemble d'apprentissage avec des experts plus ou moins agressifs) que l'on

9. Il est possible d'avoir une partition de taille supérieure à deux dans le cas d'un apprentissage multi-concepts ce qui est intéressant si le choix concerne plusieurs actions possibles.

pourra utiliser conjointement ou en fonction des situations (« est-on en avance ou en retard dans le match ? », « les adversaires sont-ils agressifs ou non ? »).

Nous avons commencé à faire une étude sur l'ouverture* de mains limites par rapport à leur nombre de points. Ces mains nécessitent un bon jugement pour prendre une décision sur leur ouverture ou non. Pour obtenir une base d'exemples assez pure, nous avons fait étiqueter des donnes de ce type générées aléatoirement par des experts via l'interface représentée dans la figure 8 où l'expert clique sur Oui s'il ouvre la main et sur Non sinon. Si l'expert ouvre la main l'exemple est placé dans E+ sinon il est placé dans E-. Pour éviter cet étiquetage manuel, on pourrait récupérer des donnes jouées par un joueur expert, faire une sélection (ex : distribution, zone de points et début de séquence pour voir quelle enchère l'expert a produit) et lancer un apprentissage de ce type dessus. Dans ce cadre, nous pouvons aussi envisager de faire un « super robot de bridge » en apprenant sur les donnes jouées par un champion pour les enchères, et par un autre expert connu pour son talent sur un compartiment du jeu de la carte.

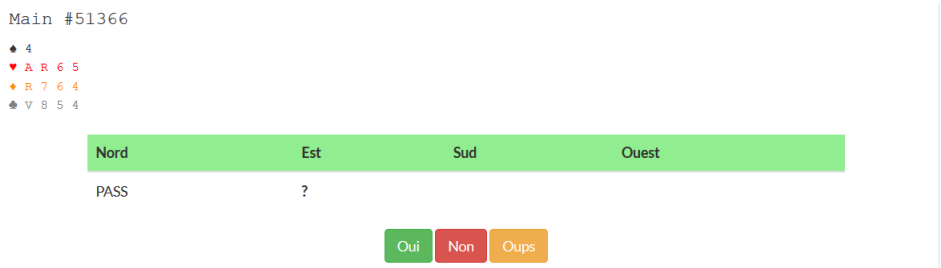


Figure 8. Interface étiquetage expert

Le traitement de cette base d'exemples est en cours à la fois pour l'approche numérique et pour l'approche symbolique.

7. Conclusion

Nous avons adapté une méthode de boosting d'IA de jeu existante aux IA de bridge et étudié son efficacité dans un cadre expérimental puis dans le cadre de championnats de robots de bridge. La méthode a donné de bons résultats, l'IA boostée avec la meilleure graine trouvée lors de nos expérimentations ayant remporté les derniers championnats du monde des robots de bridge. Nous avons également recensé un ensemble de défis permettant d'améliorer une IA de bridge en utilisant des techniques d'apprentissage récentes, les IA actuelles étant basées sur des travaux réalisés à la fin des années 1990. Parmi les techniques récentes envisagées, une appproche de type réseaux profonds paraît prometteuse sur certains compartiments du bridge. Cependant, il faut être attentif au fait que le bridge a cette particularité d'être joué à quatre sous la forme de deux paires. Au go la première étape a consisté à faire s'entraîner le robot tout seul face à des experts puis face à des versions de lui-même. Au bridge, il faudrait envisager du co-entraînement plutôt qu'un simple entraînement.

L'aspect évolution dans le temps du bridge (l'incertitude liée à la connaissance de l'état réel du monde diminue au cours de la partie) laisse à penser que l'utilisation de réseaux neuronaux récurrents pourrait être efficace. Nous pensons également qu'une approche utilisant les réseaux neuronaux profonds pourrait améliorer le comportement des robots sur la partie entame qui est cruciale au jeu de la carte.

Il reste beaucoup de chemin à parcourir pour aboutir à la création d'une IA pouvant battre des champions de bridge. Il faudrait par exemple pouvoir mettre l'adversaire à la faute (ce qui n'est pas possible avec le double dummy), gérer la coopération en flanc (le point faible des robots au jeu de la carte), adapter sa stratégie au style de l'adversaire, produire des enchères d'obstruction ayant pour but de gêner l'adversaire plutôt que de donner des informations au partenaire.

Cet article est un premier pas vers la définition d'une architecture hybride composée de modules issus de travaux récents en apprentissage automatique numérique et symbolique. Pour la partie symbolique, des travaux récents sur l'apprentissage renforcé utilisant un formalisme logique permettraient à la fois de jouer à un niveau supérieur à celui des IA actuelles et d'expliquer de manière intelligible à un humain le raisonnement effectué par le programme. Dans tous les cas, concevoir un programme de bridge expert serait un pas vers l'intelligence générale artificielle ou IA forte en opposition à l'IA appliquée.

Remerciements

Nous tenons à remercier Yves Costel qui a réalisé les expérimentations et avec qui nous avons beaucoup de plaisir à collaborer. Un grand merci aussi aux bridgeurs Jean-Baptiste Fantun, Jean-Christophe Quantin et Jean-Pierre Desmoulins pour toutes les discussions constructives que nous avons eues sur ce sujet. Enfin, cet article est dédié à Daniel Kayser pionnier français de l'intelligence artificielle qui fut mon ami, mon directeur de thèse et dont l'esprit libre restera à jamais gravé dans nos mémoires.

Bibliographie

Amit A., Markovitch S. (2006). Learning to bid in bridge. *Machine Learning*, vol. 63, n° 3, p. 287–327.

Berlekamp E. R. (1963). Program for double-dummy bridge problems: a new strategy for mechanical game playing. *Journal of the ACM (JACM)*, vol. 10, n° 3, p. 357–364.

Cazenave T., Liu J., Teytaud F., Teytaud O. (2016). Learning opening books in partially observable games: using random seeds in phantom go. *arXiv preprint arXiv:1607.02431*.

Cazenave T., Liu J., Teytaud O. (2015, Aug). The rectangular seeds of domineering. In *2015 ieee conference on computational intelligence and games (cig)*, p. 530-531.

Coulom R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, p. 72–83.

DeLooze L. L., Downey J. (2007). Bridge bidding with imperfect information. In *Computational intelligence and games, 2007. cig 2007. ieee symposium on*, p. 368–373.

Gelly S., Kocsis L., Schoenauer M., Sebag M., Silver D., Szepesvári C. *et al*. (2012). The grand challenge of computer go: Monte carlo tree search and extensions. *Communications of the ACM*, vol. 55, n° 3, p. 106–113.

Ginsberg M. (1996). Partition search. *Proceedings Of The National Conference On Artificial Intelligence*.

Ginsberg M. (2001). Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, vol. 14, p. 303–358.

Heinrich J., Silver D. (2016). Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*.

Ho C.-Y., Lin H.-T. (2015). Contract bridge bidding by learning. In *Proceedings of the workshop on computer poker and imperfect information at the twenty-ninth aaai conference on artificial intelligence.*

Liu J., Teytaud O., Cazenave T. (2016). Fast seed-learning algorithms for games. In *International conference on computers and games*, p. 58–70.

Metropolis N., Ulam S. (1949). The monte carlo method. *Journal of the American statistical association*, vol. 44, n° 247, p. 335–341.

Moravčík M., Schmid M., Burch N., Lisỳ V., Morrill D., Bard N. *et al*. (2017). Deepstack: Expert-level artificial intelligence in no-limit poker. *arXiv preprint arXiv:1701.01724*.

Munos S. G. W., Teytaud O. (2006). Modification of uct with patterns in monte-carlo go. *Technical Report RR-6062*, vol. 32, p. 30–56.

Nitti D., Belle V., De Raedt L. (2015). Planning in discrete and continuous markov decision processes by probabilistic programming. In *Joint european conference on machine learning and knowledge discovery in databases*, p. 327–342.

Paul M. (2010). Bethe. *The state of automated bridge play*.

Pepels T., Cazenave T., Winands M. H. (2015). Sequential halving for partially observable games. In *Workshop on computer games*, p. 16–29.

Schaeffer J., Burch N., Björnsson Y., Kishimoto A., Müller M., Lake R. *et al*. (2007). Checkers is solved. *science*, vol. 317, n° 5844, p. 1518–1522.

Silver D., Huang A., Maddison C. J., Guez A., Sifre L., Van Den Driessche G. *et al*. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, vol. 529, n° 7587, p. 484–489.

Smith S. J., Nau D. S., Throop T. A. (1996). Total-order multi-agent task-network planning for contract bridge. In *Aaai/iaai, vol. 1*, p. 108–113.

St-Pierre D. L., Teytaud O. (2014). The nash and the bandit approaches for adversarial portfolios. In *Computational intelligence and games (cig), 2014 ieee conference on*, p. 1–7.

Sutton R. S., Barto A. G. (1998). *Reinforcement learning: An introduction* (vol. 1) n° 1. MIT press Cambridge.

Wasserman A. I. (1970). Realization of a skillful bridge bidding program. , p. 433–444.

Yegnanarayana B., Khemani D., Sarkar M. (1996). Neural networks for contract bridge bidding. *Sadhana*, vol. 21, n° 3, p. 395–413.

Yeh C.-K., Lin H.-T. (2016). Automatic bridge bidding using deep reinforcement learning. *arXiv preprint arXiv:1607.03290*.

Annexe A. GLOSSAIRE

Sources : wikipedia.org et *http://www.lebridge.info/vocabulairedubridge.html*

A

Affranchir une couleur : jouer une couleur suffisamment souvent pour y promouvoir ses cartes restantes en cartes maîtresses.
Annonce : synonyme d'enchère.
Appel : convention au jeu de la carte qui permet d'indique au partenaire si on est intéressé par une couleur donnée ou non.
Atout : couleur préférentielle, choisie au cours des enchères, et qui permet, notamment, de couper les cartes maîtresses de l'adversaire.

B

Barrage : se dit de certaines annonces qui ont pour but d'empêcher les adversaires d'annoncer le bon contrat.
Bicolore: se dit d'une main qui possède deux couleurs longues.

C

Chicane : couleur dans laquelle un joueur ne détient aucune carte (« être chicane cœur »).
Conventions : ensemble d'enchères artificielles avec une signification bien définie ou ensemble de règles au jeu de la carte permettant par exemple de faire des appels.
Couper : jouer une carte d'atout, lorsqu'on ne possède pas la couleur demandée par l'adversaire ou le partenaire. La coupe permet (s'il n'y a pas de surcoupe) de remporter le pli.

D

Distribution : nombre de cartes qu'un joueur possède dans chaque couleur. L'ordre n'intervient pas, une distribution 5-4-3-1 est une distribution dans laquelle le joueur possède 5 cartes dans une couleur (peu importe laquelle), 4 et 3 dans deux autre couleurs et une carte seulement dans la dernière. Une distribution de ce genre est dite irrégulière. Pour simplifier on dira qu'une main régulière est une main sans singleton ni chicane.
Donne : distribution des 52 cartes aux quatre joueurs. Par extension, ensemble des phases de jeu par rapport à cette distribution.

Défausser (se) : jeter une carte d'une couleur différente de la couleur demandée, quand on ne possède pas de carte dans cette couleur.
Doubleton : couleur dans laquelle un joueur détient exactement deux cartes (« être doubleton cœur »).

E

Enchère d'essai : enchère visant à exprimer une incertitude concernant le palier du contrat idéal. Cette enchère pose une question au partenaire comme par exemple : « faut-il jouer la manche ou se contenter d'une partielle ? ».
Enchère artificielle : enchère qui par agrément, transmet au partenaire une information (différente de celle généralement admise) autre que l'intention de jouer dans la dénomination faite ou dans la dernière dénomination exprimée.
Entame : première carte jouée par le joueur qui se trouve à la gauche du déclarant.

H

Honneurs : l'As, le Roi, la Dame, le Valet et le Dix sont des honneurs, les autres cartes (du 2 au 9) sont appelées des petites cartes.

I

Impasse : technique de jeu consistant à manier une couleur de sorte à pouvoir faire un ou plusieurs plis avec des cartes de rang inférieur à celles de l'adversaire.

L

Levée : unité qui permet de déterminer le résultat du contrat. Une levée est formée par les quatre cartes que chacun des joueurs joue à son tour. La levée est remportée par le camp du joueur ayant mis la carte la plus forte.
Ligne de jeu : stratégie suivie par le déclarant pour mener à bien son contrat.
Longue : se dit d'une couleur qui possède au minimum cinq cartes.

M

Majeure : une des deux couleurs Pique ou Cœur.
Marque duplicate : compétition opposant deux équipes où chaque donne est dupliquée pour être jouée par un camp à une table et par l'autre camp ensuite.
Mineure : l'une des deux couleurs Carreau ou Trèfle.

O

Ouverture : un joueur ouvre s'il est le premier à la table à faire une annonce autre que Passe.
Ouvreur : joueur qui a ouvert .

P

Par d'une donne : résultat théorique auquel une donne devrait aboutir si tous les joueurs jouent parfaitement et ont une connaissance parfaite de la donne.

S

Sans-Atout : caractère d'un contrat dans lequel il n'y a pas d'atout.
Singleton : couleur dans laquelle un joueur ne détient qu'une seule carte.

Annexe B. Evaluation des mains

1) Le compte des points d'honneur le plus utilisé pour l'évaluation des mains est le suivant : 4 points pour un As, 3 pour un Roi, 2 pour chaque Dame et 1 point par Valet détenu. Il existe d'autres méthodes avec des valeurs différentes qui prennent par exemple en compte le nombre de 10 ou d'autres critères.

2) Le compte des points de distribution consiste à tenir compte de la distribution d'une main pour ajouter des points au compte des points d'honneur de la main. En général, on ajoute un point pour une belle couleur cinquième, un point si on a un singleton dans une couleur annexe (à moduler si on sait que c'est une couleur longue chez le partenaire), et deux points pour une chicane*.

Approche de présélection multicritère à base de traces pour la prise de décision dans les applications interactives de type jeux

Hoang Nam Ho [1], **Mourad Rabah** [1], **Samuel Nowakowski** [2], **Pascal Estraillier** [1]

1. *Laboratoire L3i, Université de La Rochelle*
   Avenue Michel Crépeau - 17042 La Rochelle Cedex 1 - France
   {hoang_nam.ho,mourad.rabah,pascal.estraillier}@univ-lr.fr

2. *LORIA, UMR 7503, Université de Lorraine*
   Campus Scientifique, BP 239, Vandoeuvre-lès-Nancy, France
   samuel.nowakowski@loria.fr

RÉSUMÉ. *La prise de décision dans les jeux est une fonctionnalité indispensable pour automatiser les jeux et les rendre plus autonomes et plus intelligents. Un algorithme de décision effectue les calculs d'optimisation sur l'ensemble des solutions possibles. Cela fait augmenter le temps de calcul et pose un problème d'explosion combinatoire lorsque l'espace de solutions est grand. Afin de surmonter ce problème, nous présentons une approche de présélection des solutions pertinentes avant d'effectuer une décision. Notre approche comporte deux étapes : i) utilisation des traces (exécutions antérieures des utilisateurs) pour identifier toutes les solutions potentielles ; ii) estimation de la pertinence, appelée utilité, de chacune de ces solutions potentielles. Nous obtenons un ensemble de solutions candidates présélectionnées qui sera utilisé comme entrée à la prise de décision. Nous expérimentons notre approche sur un prototype du jeu Tamagotchi.*

ABSTRACT. *The decision making in games is essential to make them more automated and smart. A decision algorithm performs its calculations on the set of all the possible solutions. This increases the computation time and may become a combinatorial explosion problem if we have a huge solution space. To overcome this problem, we present our work on relevant solutions preselection before making a decision. We propose a two-steps strategy: i) the first step analyzes system's traces (users past executions) to identify all the potential solutions; ii) the second step aims to estimate the relevance, called utility, of each of these potential solutions. We get a set of alternative solutions that can be used as an input to any decision algorithm. We illustrate our approach on the Tamagotchi game.*

MOTS-CLÉS : *système interactif adaptatif, traces, prédiction, utilité, décision multicritère*

KEYWORDS: *interactive adaptive system, traces, prediction, utility, multi-criteria decision making*

DOI:10.3166/RIA.31.281-305

1. Introduction

Les jeux informatisés appartiennent naturellement à la famille des applications interactives. Ces dernières sont vues comme une suite d'activités successives et d'événements réalisés entre l'utilisateur et le système (Brun, Beaudouin-Lafon, 1995). Ces activités sont les principaux composants qu'il faut mettre en œuvre en vue de disposer d'une organisation ainsi que d'une structuration des interactions, ou scénario, dans une application interactive. Nos travaux s'appuient sur l'hypothèse qu'une application interactive est contextualisée au moyen des *situations*. Une *situation* est ici vue comme un composant dans lequel les acteurs, humains ou automatisés, interagissent en utilisant des ressources locales associées à un contexte commun en vue d'atteindre un ou plusieurs objectifs identifiés (Pham *et al.*, 2015). Ainsi, l'utilisateur d'une application interactive exécute et participe à des *situations* successives jusqu'à atteindre un objectif prédéfini par le concepteur et l'exécution de l'application repose sur l'enchaînement des différentes *situations*. Nous souhaitons optimiser le processus d'enchaînement des situations composant une application interactive de type jeu par l'utilisation de l'intelligence artificielle (IA). L'IA dans les jeux comprend plusieurs domaines à traiter tels que : le dialogue automatique entre l'ordinateur et le joueur, l'aide aux diagnostics, la résolution de problèmes complexes, l'aide à la décision, etc. Dans le présent article, nous nous intéressons à l'aide à la décision dans des applications, et plus particulièrement les jeux, structurées en *situations*.

Un jeu est un terrain d'expérimentation idéal pour les aspects d'IA et en particulier du domaine d'aide à la décision. Les règles sont généralement bien définies et déterministes, ce qui permet de découper l'exécution d'un jeu en un enchaînement de situations. Avec une telle structuration, à chaque sortie de situation, nous disposons d'un vecteur d'état global représentant les attributs du système à la fin de l'exécution de la situation actuelle. En fonction de ce vecteur d'état, la logique d'enchaînement de situations va déterminer la meilleure situation à suivre parmi celles disponibles.

Le problème réside dans le choix de la situation suivante à l'issue de la situation courante tout en respectant les objectifs fixés par le concepteur du scénario et en s'adaptant au comportement de l'utilisateur. Il s'agit de concevoir des méthodes et des outils permettant d'effectuer et/ou d'accompagner le choix des situations parmi un ensemble de situations disponibles. Pour y parvenir, nous nous intéressons aux systèmes d'aide à la décision. Ces systèmes vont nous permettre d'optimiser les actions : identifier la situation qui permet de poursuivre l'exécution de l'application et respecter un ou plusieurs objectifs de l'application. La situation ainsi proposée par le système de décision est obtenue en respectant un critère déterminé. Cependant, pour une application donnée, le choix ne se base pas sur un critère unique et fait, en général, intervenir une multitude de critères. Dans ce cas, l'aide à la décision doit prendre en compte tous ces critères pour choisir une solution optimale parmi l'ensemble des solutions possibles. Ceci nous a amené à envisager des systèmes d'aide à la décision multicritère (Köksalan *et al.*, 2011). Un tel système comporte généralement cinq étapes dont les 4 dernières se répètent tout au long de l'exécution :

i) étape initiale : définition de l'objectif de l'application que l'utilisateur doit atteindre ;

ii) étape 1 : identification des critères qui nous guident dans l'accomplissement et l'évaluation de l'objectif ;

iii) étape 2 : pondération des critères pour quantifier l'importance relative entre les critères ;

iv) étape 3 : analyse des solutions possibles pour identifier les candidates et préparer la prise de décision ;

v) étape 4 : traitement des solutions candidates par les algorithmes de décision pour identifier la meilleure.

Par ailleurs, l'observation et l'évaluation de l'exécution d'une application interactive sont souvent basées sur l'analyse de grands volumes de données porteuses d'informations contextualisées, collectées pendant l'exécution, appelées *traces* (Laflaquière *et al*., 2006). Ainsi, au cours de l'exécution de l'application, la génération des traces permet de renforcer le système d'information avec des éléments qui peuvent permettre de faciliter la décision pour l'analyse et l'adaptation de la logique d'exécution. Nous proposons d'améliorer le processus de décision par l'utilisation des traces des exécutions précédentes. Un système de traces collecte les traces générées par l'utilisateur au cours de l'interaction avec le système. Puis, ces traces servent à analyser les usages dans la phase de décision (Doumat *et al*., 2010).

En combinant les aspects de décision et de traces, nous avons proposé dans (Ho *et al*., 2014) un processus général de décision multicritère à base de traces (illustré en figure 1). Nous avons repris le modèle du système d'aide à la décision classique en ajoutant les traces dans les trois dernières étapes du processus général : celle de la pondération des critères (Étape 2) (Ho *et al*., 2014), celle de la prise de décision (Étape 4) (Ho *et al*., 2015 ; 2016) et celle d'identification des solutions candidates (Étape 3), objet du présent article.

Il existe différents mécanismes de décision multicritère (Köksalan *et al*., 2011 ; Russell, Norvig, 2010). La plupart des mécanismes considèrent toutes les situations possibles pour prendre la décision, à chaque étape de décision. Le calcul s'effectue alors sur un grand ensemble de solutions. Pour surmonter ce problème, nous proposons de réduire l'espace des solutions possibles (nombre de situations) avant de lancer l'algorithme de décision (l'étape 3 dans la figure 1) en exploitant les traces d'exécution. Le présent article décrit en détail l'approche de réduction de l'espace de solutions pour la prise de décision dans un contexte à base de situations.

Pour illustrer et valider notre approche, nous avons considéré le jeu Tamagotchi comme cas d'étude. Notre prototype du jeu Tamagotchi est un jeu interactif dans lequel le joueur doit s'occuper d'un Tamagotchi, un animal virtuel que le joueur doit soigner, nourrir, faire dormir, faire jouer... En nous positionnant dans les applications interactives structurées en situations, nous contextualisons les tranches de la vie du Tamagotchi en différentes situations : *Nourrir*, *Jouer*, *Dormir*... Ainsi l'exécution de

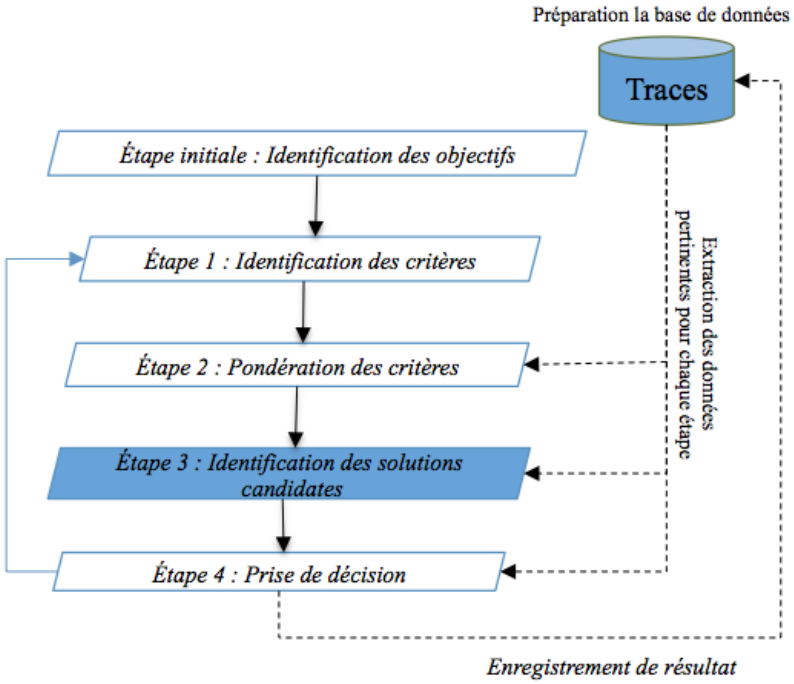


Figure 1. Processus de décision multicritère à base de traces

l'application consiste en un enchaînement de ces situations et la décision porte sur la sélection de la situation la plus appropriée à un instant donné.

Notre article est organisé comme suit. La section 2 est dédiée au positionnement de notre approche par rapport aux travaux existants autour de la réduction de l'espace de solutions possibles. Nous y abordons également les différentes techniques qui pourraient être utilisées pour résoudre notre problématique. La section 3 décrit les concepts et principes d'un système de gestion de traces. Ces traces seront utilisées dans notre approche présentée dans la section 4. La validation et l'évaluation de notre approche sur l'étude de cas Tamagotchi seront présentées dans la section 5. Enfin, nous conclurons et décrirons les perspectives dans la section 6.

2. Positionnement des travaux

La prise de décision multicritère vise à trouver une solution à partir d'un ensemble d'alternatives en synthétisant les données de différents points de vue et selon des paramètres différents provenant de différentes sources. Il existe plusieurs méthodes de décision telles que : les méthodes de pondération telles que WSW (*Weighting Sum Model*) (Triantaphyllou *et al.*, 1998) ou WPM (*Weighting Product Model*) (Triantaphyllou *et al.*, 1998), la méthode de l'utilité de multi-attributs MAUT (*Multi-Attributes Uti-*

lity Theory) (Russell, Norvig, 2010) et les méthodes de sur-classement telles que PROMETHEE (Taillandier, Stinckwich, 2011 ; Behzadian *et al*., 2010) ou ELECTRE (Corrente *et al*., 2013 ; Hatami-Marbini, Tavana, 2011). Cependant, toutes ces méthodes ne se concentrent que sur la recherche de la solution finale avec deux hypothèses qui sont : i) tous les critères ont été pondérés, ii) l'ensemble des alternatives est disponible. En ce qui concerne ces deux hypothèses, notre article vise principalement à surmonter le point ii) et de fournir une approche pour déterminer les alternatives pertinentes pour la prise de décision. Notre méthode est générale et peut être intégrée à n'importe quel algorithme de décision.

En ce qui concerne la présélection des alternatives en utilisant les traces du système, il y a quelques d'études qui tiennent compte de cet aspect. Le raisonnement à base de cas (CBR) (Riesbeck, Schank, 2013) est une méthode qui explore les traces obtenues lors des exécutions précédentes pour faire de la présélection. CBR imite le raisonnement humain et tente de résoudre de nouveaux problèmes en réutilisant des expériences antérieures. Pour chaque problème rencontré, le système a tracé les informations associées à ce problème ainsi que la solution pour le résoudre, appelé *cas*. L'idée principale de CBR est basée sur l'hypothèse que des problèmes similaires auront des solutions similaires. CBR mesure la similarité entre le problème actuel et les derniers enregistrés pour tenter d'appliquer la solution choisie à ce moment-là, qui a également été enregistrée. Si plusieurs solutions ont été envisagées dans le passé pour le même problème, les techniques d'exploration de données sont utilisées pour évaluer celle qui est la plus appropriée (Guo *et al*., 2011). Cette approche à base de cas est utilisée dans plusieurs types de jeux vidéo. Par exemple, un système CBR, intégré dans le jeu Civilisation C-evo (Sánchez-Pelegrín *et al*., 2005), est utilisé pour aider le joueur à sélectionner une action à faire quand il rencontre un problème n'ayant jamais apparu. Le système filtre les cas les plus proches en utilisant une fonction de calcul de similarité pour calculer le niveau de gain de chaque cas. Le cas ayant le gain le plus haut sera proposé aux joueurs. Dans le jeu RoboCup (championnat de football pour les robots), le CBR est appliqué pour construire des stratégies planifiées pour les robots en se basant sur les expériences passées (Karol *et al*., 2004 ; Marling *et al*., 2003). CBR a également montré ses performances dans la conception des comportements des robots dans des environnements incertains, dynamiques et en temps réel (Ros *et al*., 2009). Enfin, dans (Ontañón, Ram, 2011), les auteurs focalisent sur les techniques permettant aux utilisateurs qui n'ont pas de connaissances techniques de créer les jeux vidéo par l'utilisation CBR.

Dans (Burke, 2007 ; Cheetham, 2003), les auteurs calculent la distance pour déterminer quel élément peut être recommandé à l'utilisateur. La distance augmente lorsque la similitude est faible. Cependant, l'inconvénient majeur de cette approche est le temps de calcul. Nous devons considérer toutes les données existantes lorsque nous voulons calculer la distance pour un état. La méthode ne fournit pas un modèle général permettant d'éviter de recalculer les distances quand les nouvelles données arrivent. Dans (Dang *et al*., 2013), les auteurs utilisent la logique linéaire pour identifier toutes les alternatives possibles dans les applications à base de situations en analysant les entrées de la situation actuelle, appelées pré-conditions. Cette méthode est

très intuitive car elle effectue simplement la vérification entre les préconditions, qui font partie de la structure de la situation, et l'état actuel du système. Cependant, cette méthode nécessite une structuration en situations avec des conditions préalables et la logique linéaire ne fournit pas une mesure permettant de quantifier la pertinence des alternatives comme dans l'approche de distance ci-dessus. Cette mesure est nécessaire afin de comparer et de trier les situations disponibles.

Dans notre travail, nous voulons fournir un ensemble de situations présélectionnées pour les algorithmes de décision selon l'état actuel du système. Comme dans l'approche CBR, nous considérons l'état courant du système, nous prenons ensuite une décision en utilisant des informations enregistrées concernant les exécutions précédentes, appelé *traces*. Ces traces permettent de déterminer quelles sont les situations qui ont été choisies dans le passé dans un contexte similaire. Pour appliquer ce principe, nous intégrons un système de gestion de traces. Grâce aux traces collectées, nous calculons la probabilité d'être exécutable pour chaque situation parmi celles disponibles. Pour y parvenir, nous nous rattachons à l'apprentissage supervisé (Cornuéjols, Miclet, 2011) qui est une technique d'apprentissage automatique où l'on cherche à produire des règles et des modèles à partir d'une base de données observées et validées. Dans notre contexte, nous disposons d'une base de traces contenant les exécutions antérieures, nous appliquons dessus les techniques existantes pour construire un modèle d'apprentissage permettant de prédire les situations pertinentes. Parmi les méthodes de l'apprentissage supervisé, nous avons considéré les quatre méthodes les plus utilisées (Tan *et al*., 2006) : Bayésienne Naïve, Réseaux de neurones, k-plus proches voisins, Support Vector Machine (SVM) pour prédire quelles sont les situations pertinentes. Parmi elles, nous avons retenu la méthode Bayésienne Naïve (Ho, 2015) comme décrit dans la section 4.1.

De plus, il nous faut prendre en compte l'évaluation des situations selon plusieurs critères pour choisir celle qui peut apporter la meilleure *utilité* (Podinovski, 2014) : meilleur impact de la situation sur la progression de la valeur des critères considérés d'après les exécutions précédentes. La notion d'*utilité* a été souvent utilisée dans les systèmes d'IA et fait partie des méthodes d'apprentissage par renforcement (Sutton, Barto, 2012). Ce type d'apprentissage permet d'apprendre à partir d'expériences afin d'obtenir une récompense quantitative au cours du temps. Un système à base de l'utilité permet d'estimer le gain qu'apporte une action si elle est choisie mais ne tient pas compte du moment de l'exécution ou de l'action exécutée. Les actions potentielles sont considérées en fonction d'une variété de facteurs (Mark, 2016). Un des exemples les plus populaires d'un système d'IA à base de l'utilité est le jeu The Sims (Jeuxvidéo.com, 2008 ; Evans, 2009). Chaque action potentielle dans le jeu est évaluée en fonction d'une combinaison des besoins actuels pour estimer la capacité de satisfaction de ces besoins par l'action considérée. Chaque action est associée à une somme pondérée permettant de déterminer quelle action est « la meilleure » à l'instant donné. L'action avec le score le plus élevé sera celle choisie. Par ailleurs, dans (Dill, Mark, 2010 ; 2012) les auteurs ont montré comment utiliser la théorie de l'utilité pour modéliser la décision et l'appliquer dans les jeux vidéo. La méthode Q-learning (Watkins, Dayan, 1992) est une autre méthode typique pour l'apprentissage par renforcement.

Cependant, le temps de calcul de Q-learning augmente avec le nombre d'exemples dans la base de données. Ainsi, elle ne convient pas pour le traitement en temps réel nécessaire dans les jeux vidéos. Nous avons choisi de transposer la notion de *l'utilité* dans notre contexte à base de situations afin d'estimer l'apport que chaque situation aura si elle est choisie pour être exécutée.

En résumé, pour résoudre nos deux problèmes : i) *la construction d'un modèle de prédiction de la probabilité d'être exécutable des situations à un moment donné* et ii) *l'estimation de l'impact de chaque situation pré-sélectionnée*, nous proposons de considérer une approche hybride intégrant l'apprentissage supervisé (méthode Bayésienne) et l'apprentissage par renforcement (notion d'utilité).

Notre approche est basée sur les traces générées lors de l'interaction entre l'utilisateur et système. Avant de décrire en détail notre approche de présélection des situations candidates, nous présentons dans la section suivante notre système à base de traces.

3. Système à base de traces

Nous commençons par définir les concepts de traces et de modèle de traces (Settouti *et al*., 2009) :

– *Observé* : les observés sont des données relatives à une observation faite d'une activité ;

– *Trace* : une trace est une information ou une séquence d'informations générée par toute action relative à un objet ou un élément. Dans les systèmes informatiques, la trace est considérée comme la suite des observés ;

– *Modèle de trace* : un modèle de trace est un modèle représentant la trace d'une manière formelle. Il contient notamment les propriétés et attributs de la trace : date et heure, identifiant d'utilisateur, action réalisée, etc ;

– *Trace modélisée* : une trace modélisée ou m-trace est une trace qui est associée avec son modèle de trace ;

– *Système à base de traces* : un système à base de traces (SBT) est un système informatique permettant et facilitant l'exploitation des traces.

Lors de l'exécution, le système à base de traces fonctionne en respectant les trois sous-systèmes suivants : collecte de traces, transformation de traces et exploitation de traces. Nous décrivons sommairement chacun d'eux ci-après.

3.1. Collecte de traces

La collecte des traces permet l'observation et la récupération de plusieurs sources de données générées lors de l'exécution d'une application interactive. Elle sert à convertir ou à transformer des informations générées par l'interaction utilisateur/système pendant l'activité en une trace initiale, dite *trace première*. Dans notre contexte, nous avons identifié trois observés qui sont trois sources de traçage.

L'exécution de l'application est un enchaînement des situations. L'utilisateur participe à une situation à la fois. Pour savoir ce qui s'est passé au cours du déroulement, nous avons besoin d'observer les situations exécutées à chaque instant en analysant les interactions effectuées. C'est la première source de traçage. À chaque instant de l'exécution, nous disposons du vecteur d'état représentant l'état courant du système. Ce vecteur d'état, deuxième source de traçage, contient tous les attributs et paramètres définis par le concepteur et contribuant au fonctionnement de l'application. Quand l'utilisateur utilise l'application, il veut atteindre un certain objectif. Cet objectif est évalué par plusieurs critères. Nous ajoutons les critères en tant que troisième source de traçage. Nous cherchons à observer l'état ainsi que les valeurs associées à chaque critère, identifié au préalable, pendant l'exécution de l'application.

En nous basant sur ces trois sources de traçage, nous définissons trois types d'observés composant les traces : l'observé du système, l'observé de situation et l'observé de critère. Notre SBT collecte les données issues des interactions. Les traces collectées représentent les traces premières, notées T_P. Chaque trace première est associée à un modèle de traces permettant de recueillir les informations selon les trois observés définis. Ce modèle se compose de 3 éléments $\{V,S,C\}_t$ qui décrivent l'exécution à un moment *t* dans laquelle :

– *V* est le vecteur d'état du système, il rassemble tous les états contribuant à l'exécution de l'application ;

– *S* est la situation exécutée ;

– *C* est l'ensemble des valeurs des critères définis.

Lors de l'exécution de l'application, nous récupérons les informations selon ce modèle, nous obtenons une base des traces premières T_P. Ces traces sont conservées pour être utilisées dans les sous-systèmes suivants.

3.2. Transformation de traces

Les traces récupérées précédemment, T_P, sont des informations brutes. Nous avons besoin de les transformer pour extraire les données exploitables. Le résultat de la transformation est une base des traces transformées notée T_T. Dans notre contexte, nous avons besoin d'une base de traces transformées dans laquelle chaque enregistrement se rapporte au contexte suivant : Quel est l'état du système ? Quelle situation a été choisie pour être exécutée ? Quel est le changement dans l'évaluation des critères ?

Depuis la base de traces premières, nous extrayons des traces transformées dont chaque enregistrement contient les composants suivants :

– Ω = *{*$V \times S$*}* : est l'ensemble du vecteur d'état *V* associé à la situation exécutée *S*. Un enregistrement dans Ω a le format suivant :

$$< V = (att_1,\ att_2,...,att_m),\ S = sit_i >$$

– Δ = *{*$C_{avant} \times S \times C_{après}$*}* : l'ensemble des critères et leurs valeurs d'évaluation avant et après l'exécution de la situation *S*. Un enregistrement dans Δ est

représenté par :

$$< \{\gamma_{avant}(h)\}, S = sit_i, \{\gamma_{après}(h)\} >$$

3.3. Exploitation de traces

Dans notre SBT, la prise de décision sera abordée dans le sous-système d'exploitation de traces. Selon la figure 1, nous avons besoin d'analyser les traces pour la pondération de critères (étape 2), l'identification des solutions candidates (étape 3) et la prise de décision (étape 4). Dans la prochaine section, nous allons présenter comment nous explorons les traces pour effectuer la présélection des candidats pour la prise de décision (étape 3).

4. Approche de présélection des candidats multicritère à base de traces

Nous présentons dans cette section notre approche de présélection des situations candidates permettant de construire l'espace des solutions pour la prise de décision. Pour ce faire, nous utilisons la base de traces transformées obtenue dans la section 3.2. Notre stratégie comporte deux étapes :

1. la prédiction des situations potentielles ;
2. l'estimation de l'utilité de ces situations potentielles.

La première étape consiste à évaluer la capacité de chaque situation disponible à être potentiellement exécutable à la prochaine exécution en fonction de l'état courant du système. Nous obtenons ainsi un ensemble de probabilités associées à toutes les situations. Une *situation potentielle* est celle qui a une probabilité calculée supérieure à un seuil prédéfini et fixé par le concepteur ou l'utilisateur. Si une situation est considérée comme potentielle, nous l'ajouterons à l'ensemble des situations potentielles.

La seconde étape calcule pour chaque situation potentielle sa pertinence selon les critères définis. Cette mesure, appelée *utilité*, permet aux utilisateurs de comparer la pertinence des situations obtenues dans la première étape. La combinaison de ces deux étapes nous aide à déterminer quelles sont les situations candidates pour le processus décisionnel et de quantifier leur impact potentiel sur l'évolution des critères de l'application.

4.1. Prédiction des situations potentielles

Nous allons nous intéresser à notre stratégie de prédiction des situations potentielles à base de traces. Nous avons utilisé la partie Ω de la base de traces transformées extraite dans la section 3.2 pour construire le modèle de prédiction. Les traces obtenues sont analysées pour prédire quelles sont les situations potentielles en fonction de l'état actuel du système et d'un seuil d'acceptation défini par l'utilisateur. Pour construire le modèle de prédiction, nous avons testé les méthodes existantes de la

fouille de données comme indiqué dans la section 2 : Bayésienne Naïve (Domingos, Pazzani, 1997 ; J.Hand, Yu, 2001), Réseaux de neurones (Tan *et al.*, 2006), k plus proches voisins (Wu *et al.*, 2007) et SVM (Vapnik, 2000). Nous avons retenu l'algorithme Bayésienne Naïve parce qu'il est le plus adapté à notre problème. La comparaison détaillée de ces quatre méthodes est décrite dans (Ho, 2015). En effet, les données que nous manipulons sont de différents types (continues et discrètes). Ceci est la raison pour laquelle nous n'utilisons pas SVM bien que sa performance de prédiction soit très élevée. En raison de la petite taille de notre base de données, la méthode des k plus proches voisins ne nous permet pas d'obtenir une performance suffisante. Entre les deux méthodes restantes, Bayésienne Naïve et Réseaux de neurones, nous nous rendons compte que Réseaux de neurones doit choisir un nombre de neurones et le temps de construction du modèle de prédiction est très élevé. Cela ne peut convenir à une application interactive en temps réel de type jeu informatisé. En outre, le procédé de Bayésienne est simple et facile à mettre en œuvre. Pour conclure, nous considérons que la méthode Bayésienne Naïve est la plus appropriée pour notre cas.

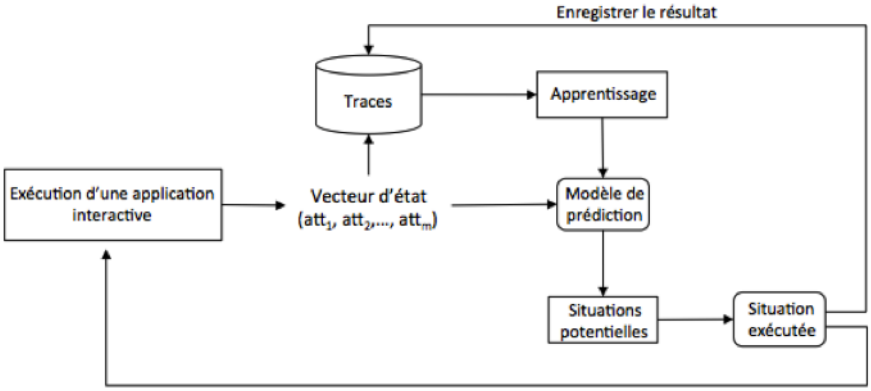


Figure 2. Processus de prédiction à base de traces des situations potentielles

Nous construisons maintenant le modèle de prédiction en appliquant la méthode Bayésienne Naïve. Le processus détaillé est présenté dans l'algorithme suivant.

Algorithme 1 – Prédiction des situations potentielles

Entrées: le vecteur d'état V, l'ensemble S de n situations et le seuil s

1: **initialiser** $Pot = \varnothing$ ▷ l'ensemble des situations potentielles
2: **pour tout** $sit_i \in S$ **faire** ▷ i de 1 à n
3: **calculer** $Prédiction(sit_i)$
4: **si** $Prédiction(sit_i) \geq s$ **alors**
5: $Pot = Pot \cup \{sit_i\}$
6: **fin si**
7: **fin pour**

Sortie: l'ensemble des situations potentielles Pot

La probabilité d'une situation *i* d'être potentiellement exécutable à partir d'un état *V = (*att_1*,* att_2*,...,*att_m*)*, noté $Prédiction(sit_i)$ est calculée par :

$$Prédiction(sit_i) = \frac{P(sit_i/V)}{\sum_{i=1}^{n} P(sit_i/V)} \quad (1)$$

Avec $P(sit_i/V)$ est la probabilité a posteriori de la situation i sachant le vecteur d'état V, elle est calculée par :

$$P(sit_i/V) = P(sit_i) \times \prod_{j=1}^{m} P(att_j/sit_i) \quad (2)$$

Dans (2), il nous faut calculer $P(att_j/sit_i)$. Elle dépend du type de donnée de l'attribut att_j. Si elle est numérique, nous supposons que ses valeurs respectent la distribution de Gauss et la probabilité de att_j sachant sit_i est alors donnée par :

$$P(att_j/sit_i) = \frac{1}{\sqrt{2\pi} \times \sigma_j^i} \times e^{\frac{-(att_j - \mu_j^i)^2}{2 \times (\sigma_j^i)^2}} \quad (3)$$

Avec μ_j^i et σ_j^i respectivement la moyenne et l'écart type de l'attribut j pour la situation i. Si la valeur de att_j n'est pas numérique, nous devrons calculer la probabilité ou la fréquence d'occurrence de att_j sachant la situation i dans la base de données pour obtenir la probabilité $P(att_j/sit_i)$.

Après avoir calculé pour chaque situation, la probabilité d'être potentiellement exécutable, nous obtenons la liste des résultats. Nous devons ensuite effectuer la vérification par la comparaison avec le seuil s. La valeur du seuil dépend du choix de l'utilisateur ou de concepteur. Le choix du seuil sera présenté dans notre cas d'étude dans la section 5. Une situation est considérée comme *situation potentielle* si sa probabilité est supérieure ou égale à s.

4.2. Estimation de l'utilité des situations potentielles

Cette étape vise à estimer, pour chaque situation potentielle, son utilité. Cette dernière représente l'impact de la situation sur la progression de la valeur des critères considérés d'après les exécutions précédentes. Pour évaluer l'utilité de chaque situation potentielle, nous avons besoin d'analyser les traces laissées dans les exécutions antérieures. Depuis la base de traces transformées, nous extrayons l'ensemble Δ qui contient des enregistrements dans lesquels sont décrits : la situation exécutée et le changement des valeurs des critères avant et après l'exécution de cette situation.

Nous commençons par le calcul de la déviation d de chaque critère. Cette valeur est la différence entre l'évaluation du critère h après ($\gamma_{après}$) et avant (γ_{avant}) avoir

exécuté une situation. Supposant que nous considérons l'enregistrement j dans la base de traces, la déviation d'un critère h est obtenue par :

$$d_h^j = \gamma_{après}^j(h) - \gamma_{avant}^j(h) \tag{4}$$

Si nous avons une base contenant q enregistrements, la déviation globale du critère h est calculée par :

$$d_h = \frac{\sum_{j=1}^{q}(\gamma_{après}^j(h) - \gamma_{avant}^j(h))}{q} = \frac{\sum_{j=1}^{q} d_h^j}{q} \tag{5}$$

Nous utilisons la déviation obtenue pour définir notre fonction d'utilité $u(d_h)$ où p_h représente le seuil d'acceptation d'une déviation d'un critère h. Ce seuil est défini expérimentalement par le concepteur de l'application. Chaque critère a un seuil différent. La valeur d'utilité d'un critère h est calculée en fonction de la déviation obtenue dans (5). Nous voulons avoir une fonction qui détermine si la valeur des critères apporte une utilité positive ou négative. Cette fonction doit se baser sur la déviation entre les valeurs pour identifier l'utilité des critères. Par conséquent, nous fixons un seuil p_h pour bien distinguer le niveau d'utilité comme montré dans (6).

$$u(d_h) = \begin{cases} -1 & \text{si} \quad d_h < 0 \\ 0 & \text{si} \quad d_h = 0 \\ \dfrac{d_h}{p_h} & \text{si} \quad 0 < d_h < p_h \\ 1 & \text{si} \quad d_h \geq p_h \end{cases} \tag{6}$$

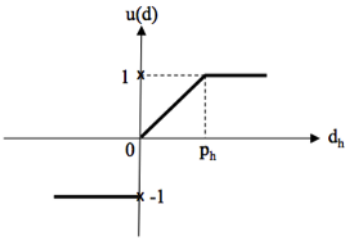


Figure 3. Fonction d'utilité

Comme illustré par figure 3, si la déviation est négative, la situation exécutée a diminué la valeur d'évaluation du critère, la valeur d'utilité correspondante sera négative. Si la déviation est supérieure à un seuil p_h, la valeur d'évaluation du critère a augmenté après l'exécution de la situation, l'utilité sera égale à 1. L'utilité sera égale à 0 si la déviation est égale à 0. Autrement, l'utilité associée vaudra $\frac{d_h}{p_h}$: plus la déviation approche du seuil, plus l'utilité approche linéairement de 1.

Algorithme 2 – Estimation de l'utilité des situations potentielles

Entrées: l'ensemble des situations potentielles Pot, l'ensemble de m critères

initialiser $Cand = \varnothing$ ▷ l'ensemble des situations candidates
pour tout $sit_i \in Pot$ **faire**
  initialiser $U = \varnothing$ ▷ l'ensemble des critères qui ont des utilités positives
  pour tout h **faire** ▷ h de 1 à m
    calculer $u(d_h)$
    si $u(d_h) \geq 0$ **alors**
      $U = U \cup \{h\}$
    fin si
  fin pour
  si $size(U) \geq K$ **alors**
    $Cand = Cand \cup \{sit_i\}$
  fin si
fin pour

Sortie: l'ensemble des situations candidates $Cand$

Nous appliquons cette approche pour estimer l'utilité des m critères de l'application pour chaque situation potentielle obtenue ci-dessus. Une situation potentielle est considérée comme *candidate* s'il existe au moins K critères sur m dont l'utilité est supérieure à 0. Dans le cas contraire, elle est *non-candidate*. Toutes les situations obtenues au bout de cette phase constituent un ensemble de candidates pour l'algorithme de décision multicritère choisi. L'algorithme 2 résume notre méthodologie.

5. Cas d'étude : jeu Tamagotchi

5.1. Description du jeu Tamagotchi

Le principe général est bien connu. Il s'agit de s'occuper d'un animal virtuel appelé Tamagotchi [1]. Nous considérons la vie du Tamagotchi à partir du début : Tamagotchi est un œuf au départ et l'utilisateur doit le faire couver afin qu'il éclose. À partir de ce moment, lorsque le Tamagotchi a besoin de quelque chose, un petit signal va apparaître pour avertir l'utilisateur afin qu'il intervienne. L'utilisateur a accès à l'état complet du Tamagotchi à travers divers indicateurs afin d'agir de façon appropriée, par exemple : donner à manger, jouer, entretenir ou éduquer... Nous avons identifié 7 situations possibles tout au long de l'exécution du jeu Tamagotchi. Elles sont :

– *Nourir* : donner à manger ou à boire au Tamagotchi ;

– *Entretenir* : nettoyer l'environnement où le Tamagotchi vit ou entretenir le Tamagotchi ;

– *Jouer* : permettre au Tamagotchi de s'amuser pour distraire le Tamagotchi ;

1. https://fr.wikipedia.org/wiki/Tamagotchi

– *Soigner* : si le Tamagotchi est malade, il a besoin d'être soigné ;

– *Dormir* : il faut faire dormir le Tamagotchi quand il est fatigué ;

– *Socialiser* : aider le Tamagotchi à avoir des amis ; on le met en contact avec d'autres Tamagotchi (gérés par l'application) ;

– *Éduquer* : apprendre les bonnes manières au Tamagotchi.

Nous avons défini le vecteur d'état du Tamagotchi qui est constitué de plusieurs attributs. Chaque attribut est une propriété décrivant une partie de son état global. Les différents attributs sont présentés dans le tableau 1.

Tableau 1. Les attributs du Tamagotchi

Attribut	**Valeur**	**Description**
satiété	[0,1]	La satiété du Tamagotchi admet une valeur de 0 (il a faim) jusqu'à 1 (il n'a pas faim)
fatigue	[0,1]	Très fatigué (0) → Pas fatigué (1)
ennui	[0,1]	Ennui maximum (0) → Ennui minimum (1)
soin	[0,1]	Cette valeur décrit le soin à accorder au Tamagotchi; plus cette valeur augmente, moins il a besoin de soin
amis	Valeur entière	Le nombre total des amis du Tamagotchi
politesse	[0,1]	Impoli (0) → Poli (1)

Figure 4. Interface du jeu Tamagotchi

De plus, nous avons défini 3 critères visant à évaluer l'objectif du jeu. Le premier critère vise à maintenir le Tamagotchi en vie jusqu'à la fin du jeu. Ce critère concerne sa *santé*. Le deuxième critère concerne la *socialisation* du Tamagotchi. Ce critère vise à faire évoluer son niveau social. Le dernier critère vise à éduquer le Tamagotchi et permet d'évaluer sa *maturité*. Les formules de calcul des valeurs des critères en fonction les valeurs des attributs sont décrites dans (Ho, 2015). Ces fonctions sont définies par le concepteur. L'interface du jeu Tamagotchi est montrée dans la figure 4.

Au début du jeu, l'utilisateur arrive à l'étape initiale, Naître. Ensuite, il enchaîne les différentes situations parmi les 7 disponibles. L'utilisateur devra notamment veiller à maintenir en vie le Tamagotchi. En effet, certaines actions peuvent mener à sa mort (s'il n'est pas bien soigné, ou s'il s'ennuie trop). Nous n'abordons pas l'ensemble des règles du fonctionnement du jeu ici, car ce n'est pas l'objet du présent l'article. Le comportement complet du jeu peut être trouvé dans (Ho, 2015). Nous décrivons dans la suite comment appliquer notre approche de présélection des situations candidates quand l'utilisateur finit une situation.

5.2. Application de l'approche de présélection des situations candidates

Notre approche de présélection est divisée en deux phases : celle d'identification des situations potentielles et celle d'estimation de l'utilité des situations potentielles. Nous disposons d'une base de traces pour effectuer le calcul (voir ci-après). Nous extrayons depuis cette base de traces premières deux bases de traces transformées (Ω et Δ comme décrit dans la section 3.2). Chacune correspond à la base de données utilisée pour chaque phase ci-dessus. La figure 5 et la figure 6 donnent un aperçu du contenu de ces deux bases.

Phase d'identification des situations potentielles

Chaque enregistrement comporte sept champs dont les six attributs du Tamagotchi (le vecteur d'état) et un dernier élément déterminant la situation exécutée : *Jouer*, *Dormir*, etc. Grâce à notre prototype, nous avons recueilli les données réelles lors de l'exécution du jeu par les utilisateurs. Notre base de traces [2] est disponible pour tester notre méthode. Quantitativement, nous avons 9315 traces qui se composent de 2261 situations *Nourrir*, de 188 situations *Entretenir*, de 559 situations *Jouer*, de 1935 situations *Soigner*, de 2217 situations *Dormir*, de 1548 situations *Socialiser* et de 607 situations *Éduquer*. Nous avons utilisé ces informations pour construire notre modèle de prédiction des situations potentielles.

Comme décrit dans la section 4.1, nous avons besoin d'un modèle de prédiction des situations potentielles parmi les situations disponibles. Pour construire ce modèle, il nous faut considérer tous les enregistrements dans la base extraite dans la figure 5 et appliquer le principe Bayésienne Naïve. Puisque les attributs dans cette base sont des valeurs numériques, il faut calculer la moyenne et l'écart type de chaque attribut.

2. Tamagotchi Traces: https://app.box.com/s/nw3jty6sfiohk9rg75uztn7yabvepkt2

L'ensemble des moyennes et des écarts types constitue notre modèle de prédiction des situations potentielles.

Relation: Tamagotchi

No.	satiété Numeric	fatigue Numeric	ennui Numeric	soin Numeric	amis Numeric	politesse Numeric	situation Nominal
1	0.97	0.42	0.93	0.31	2.0	0.1	Soigner
2	0.47	0.7	0.06	0.68	6.0	0.47	Socialiser
3	0.48	0.29	0.21	0.02	6.0	0.67	Soigner
4	0.62	0.59	0.34	0.27	7.0	0.32	Jouer
5	0.03	0.51	0.35	0.46	8.0	0.33	Nourrir
6	0.39	0.19	0.53	0.85	0.0	0.71	Dormir
7	0.46	0.34	0.2	0.35	4.0	0.49	Socialiser
8	0.56	0.83	0.42	0.8	5.0	0.95	Entretenir
9	0.24	0.76	0.57	0.78	1.0	0.24	Nourrir
10	0.38	0.3	0.29	0.78	1.0	0.65	Jouer
11	0.56	0.45	0.55	0.74	0.0	0.42	Dormir
12	0.46	0.63	0.09	0.59	0.0	0.91	Jouer
13	0.48	0.14	0.14	0.31	4.0	0.36	Dormir
14	0.97	0.52	0.47	0.8	3.0	0.61	Socialiser
15	0.07	0.22	0.66	0.32	0.0	0.39	Nourrir

Figure 5. Base de données pour la prédiction des situations potentielles

Afin d'illustrer comment identifier les situations potentielles selon le modèle de prédiction construit, nous prenons un petit exemple comme suit : le contexte du système à un instant donné est défini par le vecteur $V = (satiété = 0{,}03\,; fatigue = 0{,}26\,; ennui = 0{,}04\,; soin = 0{,}09\,; amis = 2\,; politesse = 0{,}7)$. Selon V, nous vérifions quelles sont les situations potentiellement exécutables. Nous décrivons en détail le calcul de prédiction de la situation *Nourrir*.

Deux valeurs $\mu_{satiété}^{Nourrir} = 0{,}45$ et $\sigma_{satiété}^{Nourrir} = 0{,}22$ sont respectivement la moyenne et l'écart type de l'attribut $satiété$ pour la situation *Nourrir*. Pour calculer la probabilité a posteriori de la situation *Nourrir* sachant le vecteur d'état V, noté $P(Nourrir/V)$, nous devons calculer la vraisemblance de chaque attribut du V sachant *Nourrir*, par exemple $P(satiété = 0{,}03/Nourrir)$, etc.

$$P(satiété = 0{,}03/Nourrir) = \frac{1}{\sqrt{2\pi} \times 0{,}22} \times e^{\frac{-(0{,}03+0{,}45)^2}{2\times(0{,}05)^2}} \approx 0{,}19$$

Nous procédons de la même façon pour les autres attributs et nous obtenons : $P(fatigue = 0{,}26/Nourrir)$, $P(ennui = 0{,}04/Nourrir)$, $P(soin = 0{,}09/Nourrir)$, $P(amis = 2/Nourrir)$ et $P(politesse = 0{,}7/Nourrir)$. La probabilité de la situation *Nourrir* d'après les traces est $P(Nourrir) = 2261/9315$.

$$\begin{aligned} P(Nourrir/V) = & P(Nourrir) \times P(satiété = 0{,}03/Nourrir) \times \\ & P(fatigue = 0{,}26/Nourrir) \times P(ennui = 0{,}04/Nourrir) \times \\ & P(soin = 0{,}09/Nourrir) \times P(amis = 2/Nourrir) \times \\ & P(politesse = 0{,}7/Nourrir) \approx 0{,}0012 \end{aligned}$$

Nous calculons ensuite la prédiction pour chaque situation comme décrit dans 4.1. En donnant la valeur du seuil $s = 10\%$ (ce paramètre dépend de l'application considérée). Nous obtenons l'ensemble des situations potentielles dans le tableau 2.

Tableau 2. Résultat de la prédiction des situations potentielles

Situation	Probabilié de prédiction	Résultat
Nourrir	**21,662 %**	**potentielle**
Entretenir	0,0003 %	non potentielle
Jouer	**11,128 %**	**potentielle**
Soigner	**20,3 %**	**potentielle**
Dormir	2,338 %	non potentielle
Socialiser	**16,863 %**	**potentielle**
Éduquer	**27,127 %**	**potentielle**

Selon le tableau 2, l'ensemble des situations potentielles se compose des situations suivantes : *Nourrir*, *Jouer*, *Soigner*, *Socialiser* et *Éduquer*. Pourtant, pour le moment, ce n'est que de la prédiction statistique. Cet ensemble ne prend pas en compte l'aspect multicritère. En effet, il y a des situations avec des probabilités de prédiction très élevées mais qui ne satisferont pas forcement les critères définis. Par conséquent, nous passons à la deuxième phase qui consiste à estimer l'utilité des situations potentielles identifiées.

Phase d'estimation de l'utilité des situations potentielles

Nous allons maintenant considérer la phase de calcul de l'utilité par l'analyse des traces dans la figure 6. Chaque enregistrement dans la figure 6 représente le changement des valeurs de trois critères. En détail, il contient les trois valeurs de l'accomplissement des critères avant d'exécuter la situation choisie et les trois valeurs après l'exécution de la situation choisie.

Relation: Tamagotchi

No.	santéAvant Numeric	socialisationAvant Numeric	maturitéAvant Numeric	situation Nominal	santéAprès Numeric	socialisationAprès Numeric	**maturitéAprès** Numeric
1	0.77	2.47	0.44	Socialiser	1.79	6.03	2.96
2	1.79	6.03	2.96	Soigner	0.58	6.11	4.36
3	0.58	6.11	4.36	Jouer	1.14	7.17	2.41
4	1.14	7.17	2.41	Nourrir	0.65	8.18	2.84
5	0.65	8.18	2.84	Dormir	0.9	0.27	0.53
6	0.9	0.27	0.53	Socialiser	0.95	4.1	2.25
7	0.95	4.1	2.25	Entretenir	1.77	5.21	4.87
8	1.77	5.21	4.87	Nourrir	1.21	1.29	0.37
9	1.21	1.29	0.37	Jouer	1.17	1.15	0.98
10	1.17	1.15	0.98	Dormir	1.2	0.28	0.22
11	1.2	0.28	0.22	Jouer	1.59	0.05	0.16
12	1.59	0.05	0.16	Dormir	0.79	4.07	2.15
13	0.79	4.07	2.15	Socialiser	1.82	3.24	2.02
14	1.82	3.24	2.02	Nourrir	-0.05	0.33	0.45

Figure 6. Base de données pour l'estimation de l'utilité des situations potentielles

Il faut d'abord définir les seuils pour les trois critères. Dans notre contexte, ses valeurs sont : $p_{santé} = 0{,}4$; $p_{socialisation} = 1$; $p_{maturité} = 1$. Ces seuils doivent

être définis par le concepteur ou l'utilisateur. Normalement, ces seuils devraient être définis de manière expérimentale, nous donnons un exemple qui a été expérimenté dans (Ho, 2015). Cependant, dans le cas présent, nous les avons fixé arbitrairement à titre d'exemple. Une fois les seuils définis, nous appliquons (5) et (6) pour calculer l'utilité de chaque situation potentielle. Le résultat de cette phase est résumé dans le tableau 3.

Tableau 3. Résultat d'estimation de l'utilité des situations potentielles

Situation	**Utilité du critère**			**Résultat**
	Santé	*Socialisation*	*Maturité*	
Nourrir	**1**	**0,2**	0	**alternative**
Jouer	-1	1	-1	non alternative
Soigner	**1**	0	**1**	**alternative**
Socialiser	-1	**1**	**0,6**	**alternative**
Éduquer	0	**1**	**1**	**alternative**

Nous choisissons de fixer la valeur de K à 2, c'est-à-dire qu'une situation doit avoir au moins deux valeurs d'utilité positives sur les trois critères pour être considérée comme utile. Selon l'estimation de l'utilité présentée dans le tableau 3, il y a seulement quatre situations (*Nourrir, Soigner, Socialiser* et *Éduquer*) qui font augmenter l'utilité d'au moins 2 critères en se basant sur l'analyse des traces. Si on suit l'une de ces situations, nous prédisons de globalement mieux remplir les critères de l'application. Ainsi, le résultat de notre processus de présélection des alternatives est l'ensemble des quatre situations ci-dessus.

Nous pouvons maintenant appliquer tout algorithme de décision pour calculer le choix final ou laisser à l'utilisateur le choix de sélectionner la situation à exécuter. Pour tout moment lors de l'exécution du jeu Tamagotchi, notre approche ne présélectionne que les situations candidates pour la décision, l'utilisateur a le droit de choisir n'importe quelle situation même si elle est ni utile ni potentielle.

5.3. Évaluation et discussion

Nous avons effectué des tests de performance pour valider la qualité et la pertinence de notre approche.

Comparaison des méthodes pour l'identification des situations potentielles

Tout d'abord, nous évaluons la performance de la méthode Bayésienne Naïve pour l'identification des situations potentielles en utilisant le logiciel Weka (Hall *et al.*, 2009). Dans le tableau 4, nous avons résumé le taux de prédiction correcte (mesuré avec Weka) et le temps nécessaire (en unités de temps) pour construire le modèle de prédiction en utilisant les quatre méthodes de fouille de données mentionnées à la section 2. Nous utilisons la base de traces obtenue (décrite dans la figure 5) pour effectuer l'évaluation.

Nous voyons que le taux de prédiction correct de la technique k-NN est le plus faible. Pour les 3 autres techniques, la différence du taux de prédiction n'est pas significative. Cependant, même si le taux de prédiction de Bayésienne Naïve n'est pas le meilleur comparé aux deux méthodes (Réseaux de neurones et SVM), cette différence est minime et, rapportée au temps de construction des modèles de prédiction pour ces deux méthodes, on constate qu'on obtient des résultats comparables en un temps plus court.

Tableau 4. Comparaison de performance entre quatre méthodes de prédiction : Bayésienne Naïve, k-NN, Réseaux de neurones, SVM

Méthode	**Taux de prédiction correct**	**Temps pour construire le modèle de prédiction**
Bayésienne Naïve	83,42 %	1 unité
k-NN	78,2 %	Pas besoin de modèle
Réseaux de neurones	83.8 %	108,5 unités
SVM	85.92 %	6,8 unités

Apport de notre approche de présélection dans la prise de décision

Nous avons testé l'intégration de notre approche dans les méthodes de prise de décision multicritère WSM, MAUT et PROMETHEE II pour évaluer l'apport de performance de notre approche dans une application réelle. Nous avons choisi les trois algorithmes représentant les trois familles de décision mentionnées dans la section 2 : celle de pondération (WSM), celle de l'utilité (MAUT) et celle de sur-classement (PROMETHEE II). Pour chaque algorithme, nous avons réalisé deux tests : l'un avec la décision utilisant notre approche de présélection des situations candidates et l'autre sans notre approche. Nous avons observé le temps de calcul de 20 décisions et nous avons obtenu les résultats représentés dans les figures 7, 8 et 9.

Nous observons que le temps de calcul de prise de décision avec la présélection est souvent inférieur à celui sans l'intégration de notre approche. Si nous appliquons la présélection des situations candidates avant de prendre la décision, nous pouvons réduire le nombre d'alternatives et la prise de décision n'a pas besoin de prendre en compte toutes les situations disponibles mais seulement celles qui sont pertinentes.

Toutefois, il existe certains cas où le temps de calcul avec l'intégration de notre approche est plus élevé que la décision sans notre approche, par exemple dans la figure 7 (décisions 9, 14, 17), figure 8 (décisions 4, 9, 14) et figure 9 (décisions 6, 9, 12). La raison est que, dans ces cas, toutes les situations disponibles sont considérées comme des alternatives. Par conséquent, le temps de calcul est plus élevé en raison de la charge de l'exécution de notre approche de présélection des situations candidates. Néanmoins, dans le cas général, nous pouvons remarquer de meilleurs résultats de performance de notre approche. Bien que la différence de temps de calcul pour chaque décision ne soit pas significative entre les deux stratégies, cette différence sera importante si nous considérons le temps cumulé tout au long des décisions au cours de

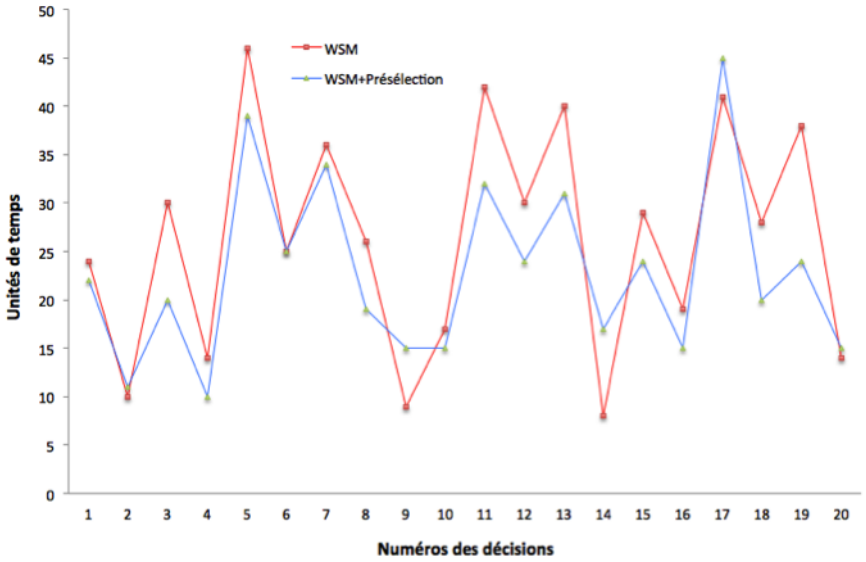


Figure 7. Comparaison du temps d'exécution de l'algorithme de décision WSM avec et sans l'approche de présélection des situations candidates

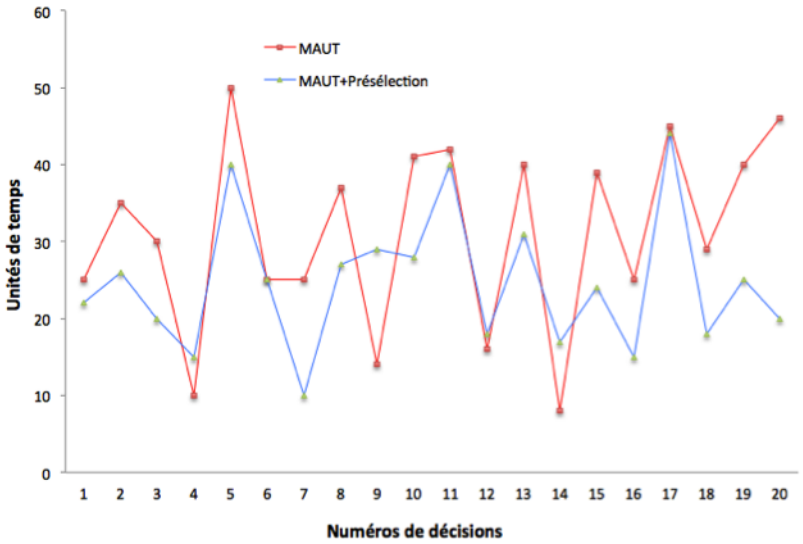


Figure 8. Comparaison du temps d'exécution de l'algorithme de décision MAUT avec et sans l'approche de présélection des situations candidates

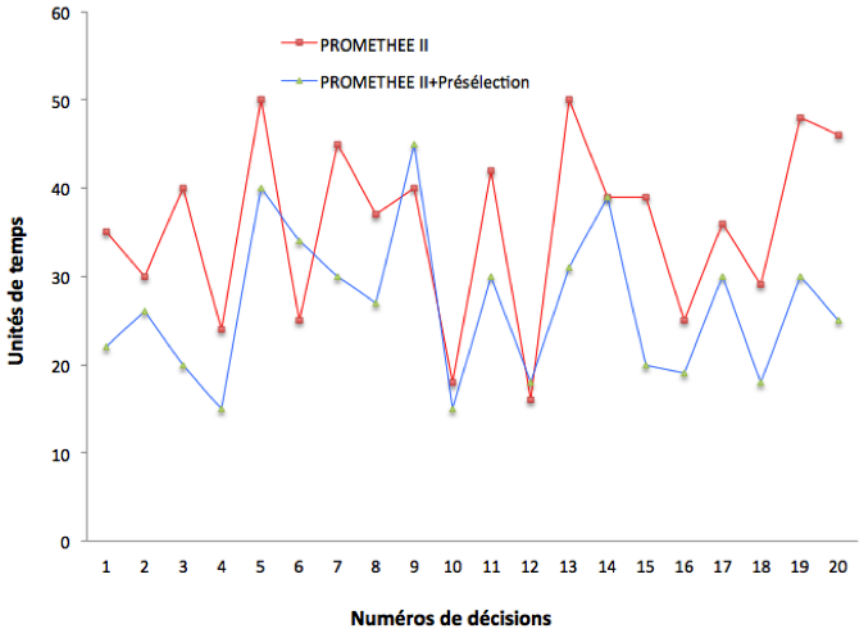


Figure 9. Comparaison du temps d'exécution de l'algorithme de décision PROMETHEE II avec et sans l'approche de présélection des situations candidates

l'exécution de l'application. En outre, cette différence augmente lorsque l'ensemble des situations disponibles de l'application augmente.

5.4. Limites

Nous avons identifié plusieurs limites à notre approche. Elle est efficace si et seulement si nous avons suffisamment de traces dans la base de données. Puisque les calculs sont basés sur l'analyse de traces générées lors des exécutions passées, la taille de la base de traces obtenue est une donnée très sensible. C'est le problème du démarrage à froid. La quantité de traces disponibles influence directement la mise en œuvre notre approche et par conséquent la pertinence des situations candidates obtenues. Pendant les premières exécutions, nous ne disposons pas suffisamment de traces pour construire le modèle de prédiction fiable. Dans ce cas, les utilisateurs doivent décider par eux-mêmes.

Une autre limite de notre méthode est le réglage des seuils s et p. Nous devons éviter de choisir des valeurs élevées, car ils représentent la limite de pertinence pour vérifier la probabilité et l'utilité. Expérimentalement, la valeur de s dans la première phase, l'identification des situations potentielles, doit être de $5\% \leq s \leq 10\%$ et la valeur de p dans la seconde phase, l'estimation de l'utilité des situations potentielles,

doit être choisie selon le type et la nature des critères pris en compte. Les choix ci-dessus s'appliquent au jeu Tamagotchi. Pour d'autres applications, nous devons effectuer plusieurs tests afin d'obtenir une valeur de seuil optimale. Le nombre de situations candidates dépend de ces valeurs.

6. Conclusion

Dans cet article, nous avons présenté une stratégie de présélection des situations candidates dans les systèmes interactifs structurées en situations. Notre approche s'appuie sur l'analyse des traces générées au cours de l'exécution. Nous avons construit un système à base de traces adapté à notre contexte et nous avons appliqué la technique bayésienne naïve afin de construire un modèle de prédiction qui nous permet d'identifier les situations potentielles en fonction de l'état actuel. Nous avons estimé ensuite l'utilité de chaque situation potentielle identifiée pour tous les critères définis. En nous basant sur ces valeurs d'utilité, nous identifions quelles sont les situations candidates pour la prise de décision. Notre approche ne modifie pas la structure des situations. Nous utilisons uniquement les exécutions passées du système, enregistrées sous forme de traces.

La contribution principale de cet article est la présélection des alternatives pour l'algorithme de décision en utilisant les traces de système afin de réduire le temps de prise de décision. Nous l'avons appliquée au cas du jeu Tamagotchi et nous avons effectué plusieurs tests pour montrer son efficacité.

Notre approche de présélection est destinée à être intégrée dans un système de décision multicritère qui pourra être appliquée pour différents types d'applications interactives telles que : les jeux sérieux, les dispositifs d'e-éducation, avec l'objectif d'optimiser le fonctionnent des algorithmes de décision. Il est à noter que notre approche pourrait être généralisée à d'autres structurations que celle des situations. En effet, la situation est un bloc de base contextualisant un certain nombre d'actions, interactions ou activités. Elle représente la granularité du découpage de l'exécution de l'application, et par conséquent pourrait être adaptée selon les besoins.

Bibliographie

Behzadian M., Kazemzadeh R. B., Albadvi A., Aghdasi M. (2010). PROMETHEE: A comprehensive literature review on methodologies and applications. *European Journal of Operational Research*, vol. 200, n° 1, p. 198–215.

Brun P., Beaudouin-Lafon M. (1995). A taxonomy and evaluation of formalisms for the specification of interactive systems. In *Proceeding of 6th International Conference on Human-Computer Interaction*, p. 197–212. Huddersfield, United Kingdom.

Burke R. (2007). Hybrid Web Recommender Systems. In Brusilovsky, Peter and Kobsa, Alfred and Nejdl, Wolfgang (Ed.), *The Adaptive Web: Methods and Strategies of Web Personalization*, vol. 4321, p. 377–408. Berlin, Heidelberg, Springer-Verlag.

Cheetham W. (2003). Global Grade Selector: A Recommender System for Supporting the Sale of Plastic Resin. In *5th International Conference on Case-Based Reasoning: Research and Development*, p. 96–106. Norway, Springer-Verlag.

Cornuéjols A., Miclet L. (2011). *Apprentissage artificiel: concepts et algorithmes.* Editions Eyrolles.

Corrente S., Greco S., Słowiński R. (2013, octobre). Multiple Criteria Hierarchy Process with ELECTRE and PROMETHEE. *Omega*, vol. 41, n° 5, p. 820–846.

Dang K. D., Pham P. T., Champagnat R., Rabah M. (2013). Linear logic validation and hierarchical modeling for interactive storytelling control. In D. Reidsma, H. Katayose, A. Nijholt (Eds.), *Advances in Computer Entertainment: 10th International Conference, ACE 2013*, vol. 8253, p. 524–527. Boekelo, The Netherlands,, Springer International Publishing.

Dill K., Mark D. (2010). Improving AI Decision Modeling Through Utility Theory (video content). In *Game Developers Conference 2010.* San Francisco, CA. http://www.gdcvault .com/play/1012410/Improving-AI-Decision-Modeling-Through

Dill K., Mark D. (2012). Embracing the Dark Art of Mathematical Modeling in AI (video content). In *Game Developers Conference 2012.* San Francisco, CA. http://www.gdcvault .com/play/1015683/Embracing-the-Dark-Art-of

Domingos P., Pazzani M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, vol. 29, n° 2-3, p. 103–130.

Doumat R., Egyed-Zsigmond E., Pinon J.-M. (2010). User Trace-Based Recommendation System for a Digital Archive. In Bichindaritz, Isabelle and Montani, Stefania (Ed.), *International Conference on Case-Based Reasoning 2010*, vol. 6176, p. 360–374. Alessandria, Italy, Springer-Verlag.

Evans R. (2009). *AI Challenges in Sims 3 - Invited talk in The Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment Conference.* Palo Alto, California. http://intrinsicalgorithm.com/IAonAI/2009/10/aiide-2009-ai-challenges-in-sims-3 -richard-evans/

Guo Y., Hu J., Peng Y. (2011). Research on CBR system based on data mining. *Applied Soft Computing*, vol. 11, n° 8, p. 5006–5014.

Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., Witten I. H. (2009). The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, vol. 11, n° 1, p. 10–18.

Hatami-Marbini A., Tavana M. (2011). An extension of the Electre I method for group decision-making under a fuzzy environment. *Omega*, vol. 39, n° 4, p. 373–386.

Ho H. N. (2015). *Décision multicritère à base de traces pour les applications interactives à exécution adaptative.* Thèse de doctorat - Université de La Rochelle, France.

Ho H. N., Rabah M., Nowakowski S., Estraillier P. (2014, août). Trace-Based Weighting Approach for Multiple Criteria Decision Making. *Journal of Software*, vol. 9, n° 8, p. 2180–2187.

Ho H. N., Rabah M., Nowakowski S., Estraillier P. (2015). Application of trace-based subjective logic to user preferences modeling. In *20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning (short papers)*, vol. 35, p. 94–105. EPiC Series in Computing.

Ho H. N., Rabah M., Nowakowski S., Estraillier P. (2016). Toward a Trace-Based PROMETHEE II Method to answer " What can teachers do? " in Online Distance Learning Applications. In *13th International Conference on Intelligent Tutoring Systems*, p. 480–484. Zagreb, Croatia.

Jeuxvidéo.com. (2008). *Les Sims passent les 100 millions.* http://www.jeuxvideo.com/news/2008/00025410-les-sims-passent-les-100-millions.htm

J.Hand D., Yu K. (2001). Idiot's Bayes - not so stupid after all? *International Statistical Review*, vol. 69, n° 3, p. 385–398.

Karol A., Nebel B., Stanton C., Williams M.-A. (2004). Case based game play in the robocup four-legged league part i the theoretical model. In D. Polani, B. Browning, A. Bonarini, K. Yoshida (Eds.), *RoboCup 2003: Robot Soccer World Cup VII*, vol. 3020, p. 739–747. Berlin, Heidelberg, Springer Berlin Heidelberg.

Köksalan M., Wallenius J., Zionts S. (2011). *Multiple criteria decision making: From early history to the 21st century*. World Scientific.

Laflaquière J., Settouti L. S., Prié Y., Mille A. (2006). Trace-Based Framework for Experience Management and Engineering. In *Proceedings of the 10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part I*, p. 1171–1178. Bournemouth, UK, Springer-Verlag.

Mark D. (2016). *Intrisic Algorithm - IA on AI.* http://intrinsicalgorithm.com/IAonAI/

Marling C., Tomko M., Gillen M., Alex D., Chelberg D. (2003). Case-based reasoning for planning and world modeling in the robocup small sized league. In *IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating*, p. 29–36. Acapulco, Mexico.

Ontañón S., Ram A. (2011). Case-based reasoning and user-generated artificial intelligence for real-time strategy games. In P. A. González-Calero, M. A. Gómez-Martín (Eds.), *Artificial Intelligence for Computer Games*, p. 103–124. New York, NY, Springer New York.

Pham P. T., Rabah M., Estraillier P. (2015). A situation-based multi-agent architecture for handling misunderstandings in interactions. *Applied Mathematics and Computer Science*, vol. 25, n° 3, p. 439–454.

Podinovski V. V. (2014). Decision making under uncertainty with unknown utility function and rank-ordered probabilities. *European Journal of Operational Research*, vol. 239, n° 2, p. 537–541.

Riesbeck C., Schank R. (2013). *Inside case-based reasoning*. Taylor & Francis.

Ros R., Arcos J. L., Mantaras R. Lopez de, Veloso M. (2009). A Case-based Approach for Coordinated Action Selection in Robot Soccer. *Artificial Intelligence.*, vol. 173, n° 9-10, p. 1014–1039.

Russell S. J., Norvig P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Settouti L. S., Prié Y., Cram D., Champin P.-A., Mille A. (2009). A Trace-Based Framework for supporting Digital Object Memories. In *1st International Workshop on Digital Object Memories (DOMe'09) in the 5th International Conference on Intelligent Environments (IE 09)*, p. 39–44. Barcelona, Spain.

Sutton R. S., Barto A. G. (2012). *Introduction to reinforcement learning* (2e éd.). MIT Press.

Sánchez-Pelegrín R., Gómez-Martín M. A., Díaz-Agudo B. (2005). A CBR module for a strategy videogame. In *1st Workshop on Computer Gaming and Simulation Environments, at 6th International Conference on Case-Based Reasoning (ICCBR)*, p. 217–226. Chicago, USA.

Taillandier P., Stinckwich S. (2011). Using the PROMETHEE multi-criteria decision making method to define new exploration strategies for rescue robots. In *IEEE International Workshop on Safety, Security, and Rescue Robotics*, p. 193–202. Kyoto, Japan.

Tan P.-N., Steinbach M., Kumar V. (2006). *Introduction to data mining*. Wesley, Pearson Addison.

Triantaphyllou E., Shu B., Sanchez S. N., Ray T. (1998). Multi-Criteria Decision Making : An Operations Research Approach. *Encyclopedia of Electrical and Electronics Engineering*, vol. 15, p. 175–186.

Vapnik V. (2000). *The Nature of Statistical Learning Theory.* New York, Springer-Verlag New York.

Watkins C. J. C. H., Dayan P. (1992). Q-learning. *Machine Learning*, vol. 8, n° 3-4, p. 279–292.

Wu X., Kumar V., Ross Quinlan J., Ghosh J., Yang Q., Motoda H. *et al*. (2007). Top 10 Algorithms in Data Mining. *Knowledge and Information Systems*, vol. 14, n° 1, p. 1–37.

WoodStock : un programme-joueur générique dirigé par les contraintes stochastiques

Frédéric Koriche, Sylvain Lagrue, Éric Piette, Sébastien Tabary

Université Lille-Nord de France CRIL - CNRS UMR 8188 Artois, F-62307 Lens
koriche@cril.fr lagrue@cril.fr epiette@cril.fr tabary@cril.fr

RÉSUMÉ. *Cet article décrit WoodStock, le premier programme-joueur générique modélisant chaque jeu issu du General Game Playing (GGP) par un réseau de contraintes stochastiques (SCSP). Chaque action jouée est décidée par la résolution de ce dernier par l'algorithme MAC-UCB. Après traduction d'une instance GDL (Game Description Language) en un réseau représentant l'état du jeu à tout temps, WoodStock résout chaque état par la maintenance d'arc-consistance (MAC) itérativement guidé par l'échantillonnage par bandit stochastique (UCB) des états suivants. À l'aide de cet algorithme, WoodStock est depuis mars 2016, le leader de la compétition continue de GGP organisée sur le serveur Tiltyard. De plus, dans sa dernière version exploitant les symétries de jeux déduites par la détection de symétries de contraintes, l'espace de recherche associé à un jeu est significativement réduit. Suite à cela, WoodStock est devenu champion lors de la compétition internationale de General Game Playing 2016 (IGGPC 2016).*

ABSTRACT. *This article describes WoodStock, the first general game player modeling each game from the General Game Playing (GGP) by a stochastic constraint network (SCSP). Each action played is decided by the resolution of this last one by the algorithm MAC-UCB. After the translation of an instance described in Game Description Language (GDL) in a network representative of the state of the game at any time, WoodStock solves each state by the maintening arc-consistency algorithm (MAC) iteratively guided by the bandit-based stochastic sampling (UCB) of the next states. Thanks to this algorithm, WoodStock is since march 2016, the leader of the GGP Tiltyard continuous tournament. Moreover, in its last version exploiting the game symmetries finding by the constraint symmetry detection, the search space associated with a game is significatively reduced. With that, WoodStock is now the GGP champion after its victory at the International General Game Playing Competition 2016 (IGGPC 2016).*

MOTS-CLÉS : *compétition internationale de* general game playing *(IGGPC), programmation par contraintes stochastiques (SCSP), échantillonnage par bandit stochastique (UCB).*

KEYWORDS: *international general game playing competition (IGGPC), stochastic constraint satisfaction problem (SCSP), bandit-based stochastic sampling (UCB).*

DOI:10.3166/RIA.31.307-336

1. Introduction

Contrairement à AlphaGo (Silver *et al.*, 2016) l'Intelligence Artificielle (IA) dédiée au *Go* remportant de nombreux succès, l'objectif du *General Game Playing* (GGP) est de concevoir des programmes-joueurs, non pas dédiés à un jeu de stratégie particulier, mais génériques de par leur capacité à jouer de manière pertinente à une grande variété de jeux. Il s'agit d'algorithmes de décisions capables de jouer à des jeux inconnus sans connaissance spécifique ne permettant donc pas l'utilisation d'heuristiques dédiées. Ils doivent plutôt être dotés de capacités cognitives de haut niveau issues du raisonnement abstrait ou de stratégies généralistes. De ce fait, le *General Game Playing* présente un défi pour l'IA englobant de nombreux thèmes comme la représentation de connaissance, la génération automatique d'heuristiques, la planification ou encore l'apprentissage automatique.

Dans le cadre GGP, les jeux sont décrits dans un langage de représentation, appelé GDL pour *Game Description Language* (Love *et al.*, 2006). Basé sur la programmation logique, la première version de ce langage capture tout jeu à information complète et parfaite. Récemment, une nouvelle version de ce langage, GDL-II (GDL *with Incomplete Information* (Thielscher, 2010)) englobe les jeux à information incomplète.

Depuis 2005, une compétition annuelle, dénommée *International General Game Playing Competition* (IGGPC (Genesereth, Björnsson, 2013)), est organisée par l'université de *Stanford* lors des conférences internationales AAAI (Genesereth, Love, 2005) ou IJCAI. Cette compétition réunie différents programmes-joueurs génériques se confrontant sur de nouvelles instances de jeux, où aucune intervention humaine n'est permise et où chaque action jouée doit être réalisée dans le temps imparti. De même, depuis 2009, *Tiltyard* organise une compétition GGP en ligne se déroulant en continue (Schreiber, 2014) et regroupant près d'un millier de programmes-joueurs génériques. Suite à ces compétitions différents programmes-joueurs génériques ont vu le jour utilisant diverses approches comme la programmation logique (Thielscher, 2005), l'*answer set programming* (Möller *et al.*, 2011), la construction automatique de fonctions d'évaluations (Clune, 2007) et ce qui fait à ce jour référence, les méthodes de type Monte Carlo (Finnsson, Björnsson, 2008 ; Cazenave, Mehat, 2010).

Dans (Koriche *et al.*, 2016), nous introduisons une nouvelle approche au *General Game Playing*, dénommée MAC-UCB, basée sur la programmation par contraintes stochastiques (SCSP) (Walsh, 2009). Expérimentalement, ce dernier s'est montré compétitif toutefois à ce jour aucun programme-joueur n'utilise actuellement cet algorithme en pratique. Ce papier présente WoodStock (*With Our Own Developper STOchastic Constraint toolKit*), notre programme-joueur générique implémentant MAC-UCB sous les contraintes de communication et de temps imposées par un tournoi GGP. WoodStock confirme les résultats de MAC-UCB sur les autres approches en devenant leader de la compétition continue depuis mars 2016. Enfin, en exploitant les symétries de jeux à l'aide de la structure graphique associée au réseau de contraintes généré pour chaque programme GDL que résout MAC-UCB, WoodStock est devenu

champion GGP 2016 en remportant la dernière compétition internationale de *General Game Playing*.

L'article est organisé de la manière suivante. Le formalisme GDL et un tour d'horizon des principales approches GGP sont réalisés en section 2 avant d'introduire la programmation par contraintes stochastiques en section 3. WoodStock est décrit en section 4 où la génération d'un SCSP équivalent à un jeu GDL est décrite avant d'introduire MAC−UCB et son implémentation. La section suivante illustre les résultats de WoodStock au cours de sa première participation à une compétition GGP mais également au cours de la compétition en continue que propose le serveur *Tiltyard*. La section 6 présente la méthode de réduction de l'espace de recherche utilisée par WoodStock à l'aide de la détection de symétries et les résultats obtenus au cours de la dernière compétition internationale de *General Game Playing* avant de conclure.

2. General Game Playing (GGP)

Le *General Game Playing* (ou souvent nommé GGP) a pris toute son ampleur en 2005 au travers d'un projet initié par le *Stanford Logic Group* [1] (Genesereth, Love, 2005). La grande variété de problèmes considérés partage une structure abstraite commune. Chaque jeu implique un nombre fini de joueurs et un nombre fini d'états, incluant un état distinct initial et un ou plusieurs états terminaux. À chaque tour de jeu, chaque joueur possède un nombre fini d'actions (aussi nommées actions légales). L'état courant du jeu est mis à jour par les conséquences simultanées de l'action de chaque joueur (qui peut être *noop* désignant l'action de ne rien faire). Le jeu commence avec un état initial et après un nombre fini de tours, se termine sur un état terminal au cours duquel un score compris entre 0 et 100 est attribué à chaque joueur.

2.1. Game Description Langage (GDL)

(Love *et al*., 2006) propose le langage *Game Description Language* (très souvent abrégé GDL) permettant d'encoder les jeux à information complète dans une forme plus compacte que leur représentation directe par un arbre de recherche. GDL propose de modéliser chaque état du jeu en utilisant la logique du premier ordre pour définir les différents composants du jeu à modéliser (actions légales, état initial et terminal, évolution du jeu, etc). (Thielscher, 2011b) montre que GDL est un langage suffisamment générique pour décrire des jeux de nombreux types.

GDL est un langage logique purement déclaratif du premier ordre où les entités (les objets) qui composent le jeu sont décrites par des termes et les propriétés sur ces entités sont décrites par des prédicats. Les atomes qui le composent sont souvent catégorisés en deux groupes distincts : la base de *Herbrand* représentant les composants booléens du jeu (les joueurs, les indices d'une ligne ou d'une colonne d'un plateau, la

1. Stanford Logic Group : http://games.stanford.edu/

valeur d'une case, etc) et les atomes d'actions représentant les actions réalisées par les joueurs. L'état d'un jeu est un sous-ensemble de la base de *Herbrand* du jeu. Tous les atomes inclus dans un état sont vrais et ceux exclus sont faux.

À chaque tour de jeu, chaque rôle peut réaliser une ou plusieurs actions légales. Une action dans un état du jeu correspond à une combinaison d'actions, une par rôle. Pour chaque action réalisée, quelques atomes de la base deviennent vrais et d'autres faux, permettant d'atteindre un nouvel état du jeu et par conséquent à de nouvelles actions légales. Dans tout jeu GDL utilisé en compétition, pour chaque état et chaque combinaison d'actions, il n'existe qu'un seul état suivant possible.

Un jeu GDL débute à l'état initial. Les différents joueurs réalisent leurs actions légales dans cet état afin d'atteindre un nouvel état. Ce processus est répété jusqu'à ce que le programme GDL atteigne un état terminal au cours duquel le jeu s'arrête et les joueurs obtiennent les scores adéquats.

GDL impose quelques mots-clés du langage pour définir une syntaxe identique quelque soit le jeu décrit. Le tableau 1 présente les dix mots-clés du langage [2].

Tableau 1. Mots-clés de GDL

Mot-clé	Description
role(j)	j est un joueur
input(j, a)	a est une action possible pour j
base(p)	p est un atome du jeu
init(p)	L'atome p est vrai à l'état initial
true(p)	L'atome p est vrai à l'état courant
does(j, a)	Le joueur j réalise l'action a à l'état courant
next(p)	L'atome p est vrai dans l'état suivant
legal(j, a)	L'action a est légale pour le joueur j à l'état courant
terminal	L'état courant est terminal
goal(j, n)	j reçoit un score de n dans l'état courant

Les différents prédicats utilisés comme mots-clés de GDL doivent répondre à certaines règles d'écriture :

– les mots-clés $role$, $base$, $input$ et $init$ utilisés dans un programme GDL sont obligatoires et doivent être décrits complètement afin de permettre la validité d'un programme GDL;

– les mots-clés $legal$, $goal$ et $terminal$ utilisés en tant que tête d'une règle, ne peuvent accepter que le mot-clé $true$ dans le corps de cette même règle ;

2. Notons que les mots-clés *input* et *base* ne sont pas présents dans l'article original présentant GDL. Ils ont été ajoutés en 2008, dans le but de rendre plus aisé une potentielle compilation de GDL vers d'autres modèles.

– le mot-clé $next$ utilisé en tant que tête d'une règle ne peut accepter que les mots-clés $true$ et $does$ dans le corps de cette même règle ;

– les mots-clés $does$ et $true$ ne peuvent pas être utilisés en tant que tête d'une règle ;

– le score possible d'un joueur défini par le mot-clé `goal` varie entre 0 et 100.

À cet ensemble de mots-clés, il est possible d'ajouter le mot-clé *not* permettant d'indiquer qu'un littéral est négatif et le mot-clé *distinct(d1,d2)* permettant d'indiquer que $d1 \neq d2$.

EXEMPLE 1. — *Le Matching pennies est un jeu simultané à deux joueurs. Chaque joueur possède une pièce. Au même moment, les deux joueurs (j_1 et j_2) retournent leur pièce sur $Pile$ ou $Face$. Une fois que les deux pièces sont retournées, le joueur j_1 remporte le jeu si les deux pièces présentent le même coté, sinon le second joueur remporte la partie. Au début du jeu, le coté de chaque pièce est assimilé à $inconnue$ (avant que la pièce ne soit dévoilée). La figure 1 correspond au programme `GDL` du Matching Pennies.*

```
% roles
role(j1).
role(j2).
% base de Herbrand
base(piece(J,C)) ← role(J), cote(C).
base(piece(J,inconnue)) ← role(J).
% actions possibles
input(J,retourne(C)) ← role(J), cote(C).
% coté d'une pièce
cote(pile).
cote(face).
% état initial
init(piece(j1,inconnue)).
init(piece(j2,inconnue)).
% actions légales
legal(J,retourne(C)) ← role(J), cote(C), true(piece(J,inconnue)).
% mis à jour de l'état du jeu
next(piece(P,C)) ← does(P,retourne(C)).
% états terminaux
terminal ← not(true(piece(P,inconnue))).
% scores
goal(J1,100) ← true(piece(J1,S)), true(piece(J2,S)).
goal(J1,0) ← true(piece(J1,S1)), true(piece(J2,S2)), distinct(S1,S2).
goal(J2,0) ← true(piece(J1,S)), true(piece(J2,S)).
goal(J2,100) ← true(piece(J1,S1)), true(piece(J2,S2)), distinct(S1,S2).
```

Figure 1. Le programme `GDL` correspondant au jeu « Matching Pennies »

Il commence par la description des joueurs $j1$ et $j2$ via l'utilisation du mot-clé role. Puis, la base de Herbrand est définie par l'utilisation du mot-clé base, ici nous définissons les différents atomes du seul prédicat ne représentant pas une action. Par la suite, les atomes d'actions sont caractérisés par l'utilisation du mot-clé input pour chaque joueur, ici seul l'atome $retourne(C)$ pour chaque joueur J est concerné.

Suite à cela, l'état initial est déclaré à l'aide du mot-clé init, ici il s'agit de le définir à l'aide des atomes $piece(j1, inconnue)$ *et* $piece(j2, inconnue)$.

Puis, par l'utilisation du mot-clé legal, nous définissons les règles logiques permettant de représenter les actions légales de chaque joueur en fonction de l'état courant défini par le mot-clé true.

Ensuite, la même chose est réalisée pour définir l'évolution du jeu par des règles logiques dont la tête est définie par le mot-clé next. Ici, on indique l'évolution des termes du prédicat $piece$ *en fonction des actions réalisées à l'état courant par chaque joueur identifié par l'utilisation du mot-clé does.*

Finalement, on définit un état comme terminal par l'utilisation du mot-clé terminal en fonction de l'état courant et les scores associés à chaque joueur par l'utilisation du mot-clé goal. Ici on associe un score de 100 *au joueur qui remporte la partie et un score de* 0 *au joueur qui la perd.*

Une extension du langage `GDL` est proposée par (Thielscher, 2010) dénommée `GDL-II` (pour *GDL with Incomplete Information*) (Thielscher, 2011a). Elle propose notamment le mot-clé *random* décrivant un joueur environnement représentant la notion de chance dans un jeu.

Le *General Game Playing* propose au travers de `GDL` un langage permettant de modéliser un grand nombre de jeux d'une grande variété. Afin de retrouver l'ensemble des jeux `GDL` valides existants, le lecteur pourra se référer à (Schreiber, 2014). Afin de motiver la recherche sur ce thème, une compétition annuelle est organisée au moment des conférences internationales $AAAI$ ou $IJCAI$, dénommée `IGGPC` (pour *International General Game Playing Competition*) (Genesereth, Björnsson, 2013).

2.2. Compétition internationale : `IGGPC`

Chaque compétition de *General Game Playing* se pratique selon le même processus au travers de matchs entre un à plusieurs joueurs selon le jeu `GDL` concerné. En début de match, chaque joueur reçoit le programme `GDL` et possède un nombre prédéfini de secondes ($startclock$), on parle de phase de pré-traitement, au cours duquel chaque joueur se prépare à jouer (analyse du jeu, développement d'une stratégie, etc). Une fois ce temps passé, le match commence. Au cours du match, chaque joueur possède un temps prédéfini ($playclock$), lui permettant de décider de sa prochaine action afin de l'envoyer. Le match se déroule tour par tour jusqu'à atteindre un état terminal du jeu `GDL` où les scores correspondants à cet état sont assignés à chaque joueur.

Au sein de la compétition, un protocole de communication est utilisé pour permettre la communication entre les différents protagonistes d'un match. Ce protocole porte le nom de `GCL` pour *Game Communication Language* au travers de connexions HTTP. Chaque joueur est à l'écoute des différents messages HTTP du *Game Manager* sur un port désigné pour la compétition. Nous détaillons brièvement les cinq types de

messages dans le tableau 2, toutefois, pour plus d'informations, le lecteur pourra se référer à (Love *et al*., 2006).

Tableau 2. GCL protocole

Message	Information
$info$	vérifie la disponibilité d'un programme-joueur
$start$	débute un match
$play$	indique les dernières actions de chaque joueur
$stop$	informe qu'un état terminal est atteint
$abort$	indique que le match est abandonné par le serveur

De plus, une compétition GGP en ligne se déroule en continue sur le serveur Tiltyard (Schreiber, 2014) où plus de $1,000$ compétiteurs s'affrontent sur plus de 150 jeux. Afin d'établir un classement entre les différents participants, le système *Agon* [3] est utilisé. Il s'agit d'une variante du classement Elo (couramment utilisé pour les échecs) au contexte GGP permettant de calculer la qualité (la compétence généraliste) de chaque joueur associée à la difficulté associée au rôle que représente le programme-joueur au cours d'un match dans chaque jeu sur la base de l'historique de l'ensemble des matchs réalisés sur le serveur. Ce système supporte les jeux à un joueur, multi-joueurs, les jeux asymétriques, les jeux à somme nulle ou non, etc.

Depuis la fondation de GDL en 2005, de nombreux programmes-joueurs ont vu le jour et certains ont été élus champions en remportant la compétition IGGPC. Le tableau 3 référence les champions annuels de GGP et les références si disponibles.

Tableau 3. Les champions GGP

Année	Programme-joueur	Référence disponible
2005	Cluneplayer	($Clune$,2007)
2006	Fluxplayer	($Schiffel, Thielscher$,2007)
2007	Cadiaplayer	($Finnsson, Bjrnsson$,2008)
2008	Cadiaplayer	($Finnsson, Bjrnsson$,2008)
2009	Ary	($Mehat, Cazenave$,2008)
2010	Ary	($Finnsson, Bjrnsson$,2011)
2011	TurboTurtle	–
2012	Cadiaplayer	($Finnsson, Bjrnsson$,2011)
2013	TurboTurtle	–
2014	Sancho	($Draper, Rose$,2014)
2015	Galvanise	($Emslie$,2015)

Le premier champion GGP est Cluneplayer, qui réalise une analyse automatique des règles du jeu dans le but de détecter différentes caractéristiques fondamen-

3. Agon : http://www.ggp.org/researchers/analysis_agonRating.html

tales d'un jeu : le gain espéré, le contrôle et la terminaison espérée d'un état donné. Chaque caractéristique identifiée est évaluée en fonction de sa corrélation avec les scores obtenus par chaque joueur. Les caractéristiques finalement sélectionnées sont combinées linéairement pour être utilisées comme fonction d'évaluations spécifique. Cette fonction associée à une recherche minimax dans l'arbre de recherche du jeu permet de décrire le mécanisme de sélection de la prochaine action à jouer.

`Fluxplayer`, le second champion emploie plusieurs éléments communs aux différents jeux `GDL` (plateau, pièces, ...) et les inclut dans une fonction d'évaluation appropriée. De plus, il utilise la logique floue pour déterminer la probabilité d'obtenir un score donné dans un état donné. Les caractéristiques atomiques sont évaluées directement alors que les structures plus complexes sont modélisées par des structures spécifiques.

Dès 2007, tous les champions suivants adoptent les méthodes de type Monte Carlo via l'utilisation de l'algorithme `UCT` (*Upper Confidence bounds applied to Trees*) (Sturtevant, 2008). Brièvement, ces méthodes se basent sur un ensemble de simulations pour choisir leurs actions, plutôt que sur la construction d'heuristiques par des fonctions d'évaluations. Chaque joueur choisit aléatoirement ses mouvements dans l'état courant jusqu'à atteindre un état terminal et enregistre les résultats obtenus sur chaque chemin parcouru. La part d'aléatoire est dirigée par la méthode `UCT` qui assure le suivi du rendement moyen de chaque combinaison état-action qui a été jouée et choisit l'action a^* à explorer dans l'état s selon :

$$a^* = argmax_{a \in A(s)} Q(s,a) + C\sqrt{\frac{ln(N(s)}{N(s,a)}}$$

où $Q(s,a)$ est la fonction d'évaluation d'état, $A(s)$ représente l'ensemble de toutes les actions possibles dans s, $N(s)$ est le nombre de fois où l'état s a été sélectionné dans les simulations précédentes, et $N(s,a)$ indique le nombre de fois où l'action a a été explorée dans l'état s.

`UCT` construit graduellement l'arbre de recherche du jeu et les paires d'état-action sont étiquetées par la moyenne des gains obtenus par simulation. Les séquences de jeux qui semblent mauvaises sont de moins en moins explorées au profit des séquences prometteuses. Ainsi, les séquences prometteuses sont développées de plus en plus vite et la profondeur de l'arbre exploré est de plus en plus importante. L'habituel dilemme exploration/exploitation de toute méthode de recherche dans un arbre est contenu par le choix du paramètre C dans la formule `UCT`.

A ce jour, les méthodes Monte Carlo restent l'état de l'art du `GGP` bien que à partir de 2011, le champion `TurboTurtle` introduit les réseaux de propositions (*propnet*) (Cox *et al.*, 2009) pour représenter les règles d'un jeu `GDL` et ainsi fortement accélérer les simulations. Avant ça, un jeu était représenté par une machine d'états où chaque état correspond à un ensemble d'atomes composés de termes et d'actions changeant d'un état à un autre. Un réseau de propositions est un graphe où les prédicats et les actions sont des nœuds plutôt que des états et où ces nœuds sont entrelacés de

nœuds représentant les connections logiques et les transitions. Un bénéfice important de cette représentation est sa densité face aux machines d'états permettant d'accroître le nombre de simulations réalisées.

(Koriche *et al.*, 2016) présente MAC−UCB, un nouvel algorithme capable d'utiliser la programmation par contraintes stochastiques (SCSP) pour déterminer ses actions. Toutefois avant d'introduire MAC−UCB, il est nécessaire de présenter le cadre SCSP.

3. Programmation par Contraintes Stochastiques (SCSP)

Le cadre CSP permet de modéliser et de résoudre un grand nombre de problèmes. Cependant, certains problèmes demandent plus de souplesse et ne sont pas modélisables par un CSP, c'est à dire uniquement sous la forme d'instanciations strictement autorisées ou strictement interdites par les contraintes. En effet, il existe un grand nombre de problèmes de décisions dont la notion d'incertitude est fondamentale.

Ainsi, dans le but de modéliser des problèmes de décision combinatoire permettant la prise en compte d'incertitudes et de probabilités, nous nous intéressons au cadre SCSP (pour *Stochastic Constraint Satisfaction Problem*) (Walsh, 2009) inspiré du problème de satisfaction stochastique (Littman *et al.*, 2001).

3.1. Définitions et Notations

Un réseau de contraintes stochastiques est un 6-uplet (X, Y, D, P, C, θ) tel que :

– X est l'ensemble fini et ordonné des n variables du problème ;

– Y est le sous-ensemble de X des variables stochastiques du problème ($Y \subset X$);

– D est l'ensemble des domaines de X ($\forall x \in X, dom(x) \in D$) ;

– P est l'ensemble des distributions de probabilités sur les domaines des variables stochastiques composant Y ($\forall x_s \in Y, P_{x_s} \in P$);

– C est l'ensemble des m contraintes du problème portant sur les variables composants X .

– θ est un seuil compris entre 0 et 1 inclus ($\theta \in [0, 1]$).

La formalisation de cette extension s'appuie sur un réseau de contraintes stochastiques où une distinction est réalisée entre deux types de variables: les variables de décisions et les variables stochastiques.

Une variable stochastique est une variable dont une distribution de probabilités est définie sur l'ensemble des valeurs définissant son domaine. Formellement, soit y une variable stochastique où $dom(y)$ représente le domaine de y. À chaque variable stochastique est associée une distribution de probabilités : il s'agit de l'ensemble des probabilités défini sur chaque valeur v du domaine de y ($v \in dom(y)$) tel que v soit affecté à y. On note $P(y = v)$ la probabilité que la valeur v soit affectée à la variable stochastique y et P_y la distribution de probabilité associée à y. Notons que

$\sum_{i=1}^{|dom(y)|} P(x = v_i) = 1$. Une variable de décision est l'appellation donnée à une variable non stochastique. C'est à dire une variable dont il n'existe pas de distribution de probabilité définie sur son domaine. Par la suite, nous notons V l'ensemble des variables de décisions d'un réseau de contraintes stochastiques $\mathcal{P} = (X, Y, D, P, C, \theta)$ tel que $V = X \backslash Y$

La portée d'une contrainte c, nommée $scp(c)$, est définie sur l'ensemble de variables X. La satisfaction d'une contrainte est la même que celle du cadre `CSP`.

Soit un sous-ensemble $U = (x_{d1}, \ldots, x_{dm}) \subseteq V$, une instanciation de U est un assignement I de valeurs $v_1 \in dom(x_{d1}), \ldots, v_m \in dom(x_{dm})$ aux variables $x_{d1}, \ldots x_{dm}$. Une instanciation I sur U est complète si $U = V$. Les assignations des variables stochastiques dans une instanciation sont appelées scénario et une probabilité leurs est associée. Soit I une instanciation d'un réseau de contraintes stochastiques $\mathcal{P} = (X, Y, D, P, C, \theta)$ où $Y = \{x_{s1}, x_{s2}, \ldots, x_{sn}\}$ ($|Y| = sn$). La probabilité associée au scénario de I notée $P(I)$ correspond au produit des probabilités d'assignation des valeurs aux variables stochastiques qui le compose. Formellement, elle est définie par : $\prod_{i=1}^{sn} P(x_{si} = v_i)$.

Une politique π pour le réseau $\mathcal{P}$ est un arbre où chaque nœud interne est étiqueté par une variable x et chaque arête est étiquetée par une valeur dans $dom(x)$. Spécifiquement, les nœuds sont étiquetés selon l'ordre X : le nœud racine est étiqueté par x_{d1}, et chaque fils d'un nœud x_{di} est étiqueté par $x_{d(i+1)}$. Les nœuds de décisions x_{di} ont un unique fils et les nœuds stochastiques x_{si} ont $|dom(x_{si})|$ enfants. Enfin, chaque feuille dans π est étiquetée par l'utilité du scénario spécifié par le chemin de la racine de π à chaque feuille. Notons qu'une politique π peut (comme une instanciation) être partielle, on note $vars(\pi)$ les variables instanciées de V du réseau de contraintes stochastiques. Une politique π couvre une contrainte c si et seulement si $scp(c) \in vars(\pi)$.

La satisfaction d'une politique est définie comme la somme de chaque valeur étiquetant chaque feuille qui la compose pondérée par la probabilité du scénario correspondant à la feuille. Une politique π d'un réseau de contraintes stochastiques $\mathcal{P} = (X, Y, D, P, C, \theta)$ est dite politique solution si sa satisfaction est supérieure ou égale au seuil θ satisfaisant par la même occasion l'ensemble des contraintes C. L'ensemble des politiques solutions d'un réseau de contraintes stochastiques $\mathcal{P}$ est noté *sols($\mathcal{P}$)*.

EXEMPLE 2. — *Soit un réseau de contraintes stochastiques $\mathcal{P} = (X, Y, D, P, C, \theta)$ avec :*

– $X = \{x_{d1}, x_{s1}, x_{d2}, x_{s2}\}$;

– $Y = \{x_{s1}, x_{s2}\}$;

– $D = \{dom(x_{d1}), dom(x_{s1}), dom(x_{d2}), dom(x_{s2})\}$ *avec* $dom(x_{d1}) = dom(x_{s1}) = dom(x_{d2}) = dom(x_{s2}) = \{0, 1, 2\}$;

– $P = \{P_{x_{s1}}, P_{x_{s2}}\}$ *avec* $P_{x_{s1}} = \{P(x_{s1} = 0) = \frac{1}{3}, P(x_{s1} = 1) = \frac{1}{3}, P(x_{s1} = 2) = \frac{1}{3}\}$ *et* $P_{x_{s2}} = \{P(x_{s2} = 0) = \frac{1}{3}, P(x_{s2} = 1) = \frac{1}{3}, P(x_{s2} = 2) = \frac{1}{3}\}$;

– $C = \{c_1, c_2, c_3\}$ *avec* $scp(c_1) = \{x_{d1}, x_{d2}\}$ *où* $c_1 : x_{d1} = x_{d2}$, $scp(c_2) = \{x_{d1}, x_{s1}\}$ *où* $c_2 : x_{d1} + x_{s1} > 1$ *et* $scp(c_3) = \{x_{d2}, x_{s2}\}$ *où* $c_2 : x_{d1} + x_{s2} > 1$ *;*
– $\theta = \frac{1}{3}$.

La figure 2 représente une politique pour le problème SCSP $\mathcal{P}$. *La politique représente la décision d'assigner la valeur* 1 *aux variables* x_{d1} *et* x_{d2}. *Dans cette politique, la contrainte* $c_1 : x_{d1} = x_{d2}$ *portant uniquement sur les variables de décisions est satisfaite. Ici, les neuf chemins entre chaque feuille et la racine représentent les neuf scénarios possibles. Par exemple, le scénario où la valeur* 0 *est assignée aux deux variables stochastiques* x_{s1} *et* x_{s2} *est le chemin entre la feuille la plus à gauche et la racine de chaque politique. Ce scénario ne satisfait pas les contraintes* $c_2 : x_{d1} + x_{s1} > 1$ *et* $c_3 : x_{d2} + x_{s2} > 1$, *la feuille correspondante est alors étiquetée par la valeur* 0. *Contrairement au scénario représenté par le chemin entre la feuille la plus à droite et la racine où la valeur* 2 *est assignée aux deux variables stochastiques qui est étiquetée par la valeur* 1.

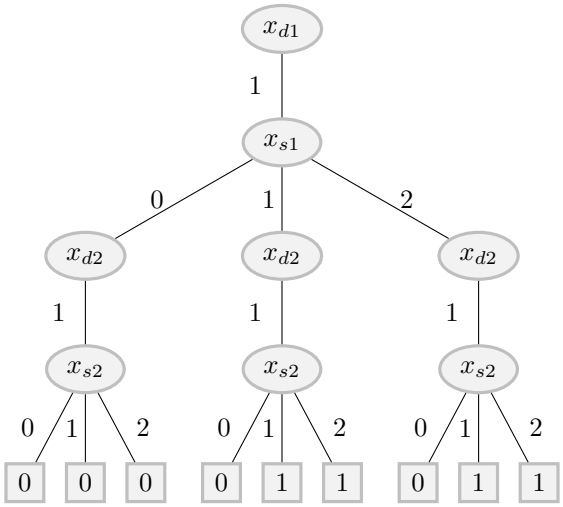


Figure 2. Une politique solution pour le SCSP $\mathcal{P}$

La satisfaction associée à chaque politique est égale à la somme de chacune des valeurs étiquetant les feuilles pondérées par la probabilité du scénario qui correspond. Dans cet exemple, la probabilité est uniforme pour chaque scénario, elle est correspond à $\frac{1}{3} \times \frac{1}{3} = \frac{1}{9}$. *Ainsi la satisfaction* s *de la politique est* $\frac{4}{9}$. *nous avons* $s > \theta$ *signifiant que la politique est une solution pour le problème* $\mathcal{P}$.

Il est possible de représenter un réseau de contraintes stochastiques par un m-SCSP (un SCSP à m niveaux) où chaque niveau représente un sous-ensemble de X nommé $X_i = V_i \cup Y_i$ tel que $X = \bigcup_{i=1}^{m} X_i$. Dit autrement $X = \{V_1, Y_1, V_2, Y_2, \ldots, V_m, Y_m\}$.

EXEMPLE 3. — *L'exemple 2 correspond à un* 2-SCSP *où* $X_1 = \{x_{d1}, x_{s1}\}$ *et* $X_2 = \{x_{d2}, x_{s2}\}$.

Notons que le SCSP à un seul niveau (1-SCSP) est nommé μSCSP (pour micro-SCSP) dans cet article. Un micro-SCSP est un SCSP $\mathcal{P} = (X, Y, D, C, P, \theta)$ dont l'ensemble X composant les n variables est ordonné strictement par l'ensemble des dn variables de décisions composant V puis par l'ensemble des sn variables stochastiques composant Y. Dit autrement, si $V = \{x_{d1}, x_{d2}, \ldots, x_{dn}\}$ et $Y = \{x_{s1}, x_{s2}, \ldots, x_{sn}\}$ alors $X = V \cup Y = \{x_{d1}, x_{d2}, \ldots, x_{dn}, x_{s1}, x_{s2}, \ldots, x_{sn}\}$.

Un μSCSP peut être satisfait si il existe un ensemble d'assignations pour les variables de décisions tel qu'il existe des scénarios possibles dont la somme des probabilités de chaque scénario est supérieure ou égale au seuil attendu.

Si on étend la résolution d'un μSCSP à un m-SCSP, on peut comprendre que la résolution d'un tel problème est de déterminer un ensemble d'assignations pour les variables de décisions composant le premier niveau (μSCSP_1) où soit un ensemble de valeurs aléatoires données pour les variables stochastiques, il est possible de déterminer un ensemble d'assignations pour les variables de décisions composant le second niveau (μSCSP_2), etc ... jusqu'au dernier niveau (μSCSP_m), tel qu'il existe des scénarios possibles dont la somme des probabilités de chaque scénario est supérieure ou égale au seuil θ attendu.

Il est important de préciser que les contraintes d'un SCSP porte sur l'ensemble des différents niveaux le composant, il n'est donc pas possible de décomposer l'ensemble des contraintes C comme il est possible de le faire pour l'ensemble X des variables.

4. WoodStock

Dans cette section, nous présentons WoodStock (*With Our Own Developper STOchastic Constraint toolKit*), le premier programme-joueur générique dirigé par les contraintes dans un tournoi GGP.

Comme vu en section 2.2, un programme-joueur doit pouvoir communiquer avec le *game manager* avant de s'intéresser au jeu lui-même. À chaque tour, pour obtenir l'état courant et communiquer les actions choisies de WoodStock, nous avons développé un joueur-réseau basé sur le protocole GCL dénommé *spy-ggp* [4]. Il utilise uniquement la librairie standard *Python* et n'a besoin que de l'interpréteur *Python3* pour fonctionner. Il est possible de lier à toute librairie dynamique (.so, .dll, .dylib, ... en accord avec le système utilisé) modélisant la partie stratégie du joueur. Cette librairie peut être écrite en tout autre langage (C, C++, C#, Go, OCaml, ...). Depuis sa première utilisation en compétition GGP, il s'est démontré robuste sans ne jamais provoquer d'erreur.

Le langage C++ a été choisi pour implémenter la partie stratégique de WoodStock afin de bénificier du paradigme proposé par la programmation orientée objet (POO) et de sa bonne capacité de calcul. L'utilisation de la POO permet d'obtenir une concep-

4. *spy-ggp*: https://github.com/syllag/spy-ggp disponible sous Licence GNULGPL3 / CeCILL-C.

tion élégante permettant de faciliter la maintenance et l'évolution du joueur au fil du temps.

La partie stratégique est composée de trois parties : (1) l'étape de traduction ; (2) la phase de résolution ; (3) la phase de simulation.

4.1. De GDL à SCSP

Au cours du temps de pré-traitement alloué (*start clock*), WoodStock réalise la traduction d'un programme GDL impliquant k joueurs en un SCSP dans le but que ce dernier puisse être utilisé le plus efficacement en compétition. Tout d'abord, suite à la réception du programme GDL, WoodStock génére un *template* dénommé μSCSP_t représentatif d'un tour quelconque du jeu GDL. Ce patron est une traduction du programme GDL sans y inclure les règles construites via le prédicat « init » modélisant l'état initial du jeu.

Dans un premier temps, nous appliquons la méthode de Lifschitz et Yang (Lifschitz, Yang, 2011) sur le programme GDL afin d'éliminer les fonctions et ainsi de simplifier la traduction. Cette méthode inclut deux étapes : l'*aplanissement* permettant de réduire les termes imbriqués, puis la phase d'*élimination* qui remplace ces fonctions par des prédicats. L'*aplanissement* est spécifiée comme suit : pour chaque symbole $\mathtt{f}$ qui apparaît dans P, nous construisons un équivalent $\mathtt{P_f}$, dénommé le $\mathtt{f}$-*aplanit*, dans lequel chaque occurrence d'un terme de la forme $\mathtt{f}(\mathtt{t_1}, \cdots, \mathtt{t_k})$ est transformé par l'égalité $\mathtt{f}(\mathtt{t_1}, \cdots, \mathtt{t_k}) = \mathtt{Z}$, où Z est une variable de renommage. Par exemple, si le programme GDL P contient $\mathtt{legal}(\mathtt{random}, \mathtt{roll}(\mathtt{c}))$, alors $\mathtt{P_f}$ est remplacé par la conjonction $\mathtt{legal}(\mathtt{random}, \mathtt{Z}), \mathtt{roll}(\mathtt{c}) = \mathtt{Z}$. Par l'aplanissement de toutes les fonctions f de P, plus aucun terme imbriqué n'apparaît. Notons que cette phase est polynomialement bornée par le nombre de fonctions et la profondeur des termes. L'étape d'élimination remplace chaque symbole de fonction f d'arité k par un symbole de relation F d'arité $k+1$. Ainsi, chaque égalité de la forme $\mathtt{f}(\mathtt{t_1}, \cdots, \mathtt{t_k}) = \mathtt{Z}$ est remplacée par l'atome $\mathtt{F}(\mathtt{t_1}, \cdots, \mathtt{t_k}, \mathtt{Z})$. Finalement, la règle $(\exists!\mathtt{Z})\mathtt{F}(\mathtt{t_1}, \cdots, \mathtt{t_k}, \mathtt{Z})$ est ajoutée au programme $\mathtt{P}'$ afin d'assurer l'équivalence des modèles entre P et $\mathtt{P}'$.

Par la suite, WoodStock génère chaque variable du programme SCSP: une variable $terminal$ modélisant si un état est terminal ou non, k variables $score_j$ afin de modéliser le score de chaque joueur j, k variables $action_j$ pour modéliser l'action de chaque joueur j et deux variables pour chaque fluent du jeu, respectivement une au temps courant t et une seconde au temps suivant $t+1$.

La prochaine étape est l'extraction des domaines. Le domaine de $terminal$ est booléen et le domaine de chaque variable $score_j$ est modélisé par la valeur entière apparaissant dans chaque règle « goal » correspondant à chaque joueur j. Finalement pour chaque variable $action_j$ et pour chaque variable modélisant un fluent au temps t ou $t+1$, WoodStock exploite les conditions d'une compétition GGP actuelle imposant l'utilisant des règles « input » et « base » composant un programme GDL. Ces derniers permettant de générer rapidement l'univers de Herbrand, il n'est pas néces-

saire de calculer l'ensemble des combinaisons des constantes pour obtenir le domaine de chaque variable « fluent ». Toutes les variables générées sont des variables de décisions sauf la variable correspondant à l'action de l'environnement *random* si celui-ci est utilisé. La distribution de probabilité associée à l'unique variable stochastique est uniforme sur l'ensemble des actions possibles de l'environnement composant son domaine. Enfin, le seuil θ est fixé à 1 pour que seuls les scénarios composés d'actions légales, soient conservés dans l'ensemble des solutions.

Tableau 4. Les règles de réécriture entre `GDL` *et les portées des contraintes*

Prédicat `GDL`	**Portée de la contrainte**
$true(f(\ldots))$	$\{f_t\} \in scp(C_t)$
$does(j, a(\ldots))$	$\{action_{j,t}\} \in scp(C_t)$
$legal(j, a(\ldots))$	$\{action_{j,t}\} \in scp(C_t)$
$next(f(\ldots))$	$\{f_{t+1}\} \in scp(C_t)$
$goal(j, N)$	$\{score_{j,t}\} \in scp(C_t)$
$terminal$	$\{terminal_t\} \in scp(C_t)$

La dernière étape est la génération de chaque contrainte. `WoodStock` commence par créer une contrainte pour chaque règle `GDL` en associant à chaque contrainte sa portée correspondante suite aux règles de réécriture détaillées dans le tableau 4. Quant aux relations des contraintes, elles sont restreintes uniquement aux termes présents dans chaque règle `GDL` correspondante. La présence des mot-clés « distinct » et « not » sont utilisés pour effectuer un premier filtrage de chaque relation. Les contraintes obtenues sont en extension et modélisées par des contraintes tables.

EXEMPLE 4. — *La figure 3 illustre un $\mu SCSP_t$ correspondant au programme* `GDL` *au temps t du « Matching Pennies », illustré dans l'exemple 1. Notons que les variables $action_{j1,t}$ et $action_{j2,t}$ sont assimilées respectivement à $retourne_{j1,t}$ et $retourne_{j2,t}$ pour des raisons de lisibilité afin de ne pas répéter le nom de l'action dans le domaine de chacune de ces variables. De même, la variable $terminal_t$ n'est pas exprimée dans les relations composant les contraintes goal afin de clarifier la lecture. Rappelons que si $terminal_t$ est faux alors quelque soit les valeurs des variables fluents $score_t$ est égale à 0. Les notations suivantes sont utilisées : P = $pile$; F = $face$; I = $inconnu$; V = $vrai$; W = $faux$.*

Premièrement, les trois variables $terminal_t$, $score_{j1,t}$ et $score_{j2,t}$ représentent respectivement la fin du jeu et les scores possibles de $j1$ et $j2$. Ces variables sont extraites des mots-clés $terminal$ et $goal(J, S)$. Le domaine associé à $terminal_t$ est booléen et celui des deux autres variables correspond aux différentes valeurs S possiblement impliquées dans $goal(J, S)$.

Les états du jeu au tour t et $t + 1$ sont représentés par l'ensemble des variables dérivées de par le mot-clé next. Le domaine de ces variables correspond aux valeurs possibles du fluent C utilisé dans l'exemple. Puis, on utilise le mot-clé legal pour générer les variables $retourne_{j1,t}$ et $retourne_{j2,t}$.

Une contrainte terminale relie la variable $terminal_t$ aux deux variables représentant les pièces. Le jeu se termine quand toutes les pièces ne sont plus assignées à la valeur « inconnue ». De la même manière, la contrainte goal relie les différents côtés de chacune des pièces aux scores associés à $j1$ et $j2$. Finalement les contraintes next permet au jeu de passer d'un état t à un autre état $t+1$ dépendant des actions choisies par les joueurs (variables $retourne_{j1,t}$ et $retourne_{j2_t}$).

Variable	Domaine
$\texttt{piece}_{j1,t}$	{F, P, I}
$\texttt{piece}_{j2,t}$	{F, P, I}
$\texttt{terminal}_t$	{V, W}
$\texttt{score}_{j1,t}$	{0, 50, 100}
$\texttt{score}_{j2,t}$	{0, 50, 100}
$\texttt{retourne}_{j1,t}$	{F, P}
$\texttt{retourne}_{j2,t}$	{F, P}
$\texttt{piece}_{j1,t+1}$	{F, P, I}
$\texttt{piece}_{j2,t+1}$	{F, P, I}

Variables et domaines

$\texttt{piece}_{\texttt{j1,t}}$	$\texttt{pieces}_{\texttt{j2,t}}$	$\texttt{terminal}_{\texttt{t}}$
F	F	T
F	P	T
P	F	T
P	P	T
I	F	W
I	P	W
F	I	W
P	I	W

Contrainte *terminal*

$\texttt{piece}_{j1,t}$	$\texttt{retourne}_{j1,t}$
I	P
I	F
$\texttt{piece}_{j2,t}$	$\texttt{retourne}_{j2,t}$
I	P
I	F

Contraintes *legal*

$\texttt{piece}_{\texttt{j1,t}}$	$\texttt{piece}_{\texttt{j2,t}}$	$\texttt{score}_{j1,t}$
F	F	100
P	P	100
F	P	0
P	F	0
$\texttt{piece}_{\texttt{j1,t}}$	$\texttt{piece}_{\texttt{j2,t}}$	$\texttt{score}_{j2,t}$
F	P	100
P	F	100
F	F	0
P	P	0

Contraintes *goal*

$\texttt{piece}_{j1,t}$	$\texttt{retourne}_{j1,t}$	$\texttt{piece}_{j1,t+1}$
I	F	F
I	P	P
$\texttt{piece}_{j2,t}$	$\texttt{retourne}_{j2,t}$	$\texttt{piece}_{j2,t+1}$
I	F	F
I	P	P

Contraintes *next*

Figure 3. $\mu SCSP_t$ encodant le « Matching Pennies »

`WoodStock` utilise `XCSP 3.0`[5], un format basé sur `XML` pour représenter les instances obtenues dans le but de fournir de nouvelles instances `SCSP` et d'en extraire facilement la partie `CSP`.

5. `XCSP 3.0` : xcsp.org

4.2. MAC-UCB

Dans (Koriche *et al.*, 2016), la modélisation SCSP obtenue est identifiée dans un fragment SCSP où chaque contrainte ne porte que sur un seul et unique μSCSP composant le réseau. Ce fragment est exploité par WoodStock en utilisant la technique de résolution MAC-UCB.

Comme indiqué auparavant, le réseau de contraintes stochastiques d'un programme GDL est une séquence de μSCSPs, chacun associé à un tour de jeu. Pour chaque $\mu\text{SCSP}_t \in \{0, \ldots, T\}$, MAC-UCB recherche l'ensemble des politiques solutions en décomposant le problème en deux parties: un CSP classique et un μSCSP (plus petit que l'original). La première partie est résolue à l'aide de l'algorithme MAC (Sabin, Freuder, 1994) et la seconde partie grâce à l'algorithme SFC dédié au cadre SCSP (Walsh, 2009). Par la suite, une série d'échantillonnages avec borne de confiance est réalisée pour simuler l'utilité attendue de chaque politique solution de chaque μSCSP_t.

Suite à la génération du *template* μSCSP_t, quelques méthodes de pré-traitement ne demandant que très peu de temps y sont appliquées afin de rendre la phase de résolution plus efficace. Les contraintes possédant la même portée sont fusionnées en une seule contrainte en unifiant les relations. Puis, toutes les variables universelles, c'est à dire les variables dont toutes les valeurs sont comprises dans l'ensemble des solutions, sont supprimées.

Afin d'obtenir un μSCSP_t à un temps donné t, WoodStock utilise un injecteur. Au temps $t = 0$ représentant l'état initial, l'injecteur correspond à un ensemble de contraintes unaires générées à l'aide des règles « init ». À tout autre temps $t \neq 0$, l'injecteur est construit avec les solutions du micro SCSP correspondant à l'état à l'origine du nouvel état. Grâce à celui-ci, il est possible de projeter la relation de chaque contrainte unaire ajoutée sur le domaine de chaque variable apparaissant dans la portée sur l'ensemble du réseau de contrainte en le restreignant à la seule valeur autorisée par le tuple de la contrainte associée.

Le μSCSP obtenu est alors assez petit pour appliquer l'algorithme SAC (Debruyne, Bessière, 1997) sur ce dernier dans les délais imposées par une compétition GGP. Suite à cela, les valeurs inconsistantes sont supprimées du domaine des variables. C'est pourquoi, seules les actions légales représentées par les valeurs consistantes des variables $action_j$ de chaque joueur j sont conservées et garantissent la légalité des actions jouées par WoodStock au cours de l'ensemble du jeu.

4.2.1. Phase de résolution : MAC

Après la réalisation de ces techniques de pré-traitement, l'objectif de la phase de résolution est d'énumérer l'ensemble des politiques solutions du μSCSP, dont certaines peuvent mener à une solution optimale.

L'algorithme de *Forward Checking* (SFC) adapté à ce cadre (Walsh, 2009) utilisé seul n'est pas suffisant. En effet, dans le cadre d'un μSCSP impliquant un nombre

de contraintes important, SFC n'est pas assez efficace suite à sa faible capacité de filtrage. Par conséquent, l'algorithme MAC−UCB (Koriche *et al.*, 2015) une technique de résolution adaptée au SCSP à un niveau est utilisé par notre programme-joueur.

WoodStock sépare le μSCSP $\mathcal{P}$ en un CSP $\mathcal{P}'$ modélisant la partie dure composée uniquement des contraintes dures de $\mathcal{P}$ (voir algorithme 1) et en un μSCSP $\mathcal{P}''$ modélisation la partie chance contenant uniquement les contraintes de chances de $\mathcal{P}$ (voir algorithme 2). Les politiques solutions du μSCSP sont alors identifiées en combinant les solutions du CSP $\mathcal{P}'$ avec les solutions du μSCSP $\mathcal{P}''$.

Algorithme 1 : *partie_dure*

Données : μSCSP $\mathcal{P} = (X, Y, D, C, P, \theta)$
Résultat : CSP $\mathcal{P}' = (X', D', C')$
$\mathcal{P}' = (X' \leftarrow \emptyset, D' \leftarrow \emptyset, C' \leftarrow \emptyset)$
pour $c \in C$ **faire**
  si $\nexists x_s \in scp(c) \mid x_s \in Y$ **alors**
    $C' \leftarrow C' \cup \{c\}$
    pour $x_d \in scp(c)$ **faire**
      $X' \leftarrow X' \cup \{x_d\}$
      $D' \leftarrow D' \cup \{dom(x_d)\}$
    fin
  fin
fin
retourner $\mathcal{P}'$

Algorithme 2 : *partie_chance*

Données : μSCSP $\mathcal{P} = (X, Y, D, C, P, \theta)$
Résultat : μSCSP $\mathcal{P}'' = (X'', Y'', D'', C'', P'', \theta)$
μSCSP $\mathcal{P}'' = (X'' \leftarrow \emptyset, Y'' \leftarrow \emptyset, D'' \leftarrow \emptyset, C'' \leftarrow \emptyset, P'' \leftarrow \emptyset, \theta)$
pour $c \in C$ **faire**
  si $\exists x_s \in scp(c) \mid x_s \in Y$ **alors**
    $C'' \leftarrow C'' \cup \{c\}$
    pour $x \in scp(c)$ **faire**
      si $x \in Y$ **alors**
        $Y'' \leftarrow Y'' \cup \{x\}$
        $P'' \leftarrow P'' \cup \{P_x\}$
      fin
      sinon
        $X'' \leftarrow X'' \cup \{x\}$
      fin
      $D'' \leftarrow D'' \cup \{dom(x)\}$
    fin
  fin
fin
retourner $\mathcal{P}''$

Tout d'abord, examinons la résolution de $\mathcal{P}''$. Pour des instances GDL décrivant des jeux à information imparfaite, $\mathcal{P}''$ inclut au plus une unique contrainte (stochastique) capturant la règle de transition impliquée par le joueur environnement (*random*). Par

conséquent, $\mathcal{P}''$ peut être facilement résolu par SFC adapté au cadre des SCSP à un niveau. Par la suite, l'ensemble de politiques solutions obtenues par SFC sur $\mathcal{P}''$ est encodé par une contrainte dure c_f, dénommée contrainte de faisabilité, où $scp(c_f)$ contient l'ensemble des variables de décisions de $\mathcal{P}''$ et $rel(c_f)$ est l'ensemble des tuples correspondant aux assignations des variables de décisions qui font parties d'au moins une politique solution de $\mathcal{P}''$. Formellement, soit $sols(\mathcal{P}'')$ l'ensemble des politiques solutions de P'' alors :

- $scp(c_f) = vars(\pi)$ tel que $\pi \in sols(\mathcal{P}'')$;
- $rel(c_f) = (I_0, \ldots, I_i, \ldots, I_q)$ tel que $I_i \models \pi_i \in sols(\mathcal{P}'')$ avec $|sol(\mathcal{P}'')| = q$.

Désormais, intéressons nous à la résolution du CSP $\mathcal{P}'$. Ici, l'algorithme classique MAC (Sabin, Freuder, 1994) est utilisé pour énumérer toutes les solutions de $\mathcal{P}'$. Dans le but de tenir compte des solutions identifiées dans $\mathcal{P}''$, nous ajoutons simplement la contrainte de faisabilité c_f aux contraintes C' de $\mathcal{P}'$. Les solutions obtenues par MAC sur $\mathcal{P}' = (X', D', C' \cup \{c_f\})$ sont identiques à l'ensemble des solutions du μSCSP $\mathcal{P}$ original. MAC exploite la propriété de consistance d'arc avec pour objectif de filtrer efficacement les politiques non satisfaisantes du CSP $\mathcal{P}'$. La stratégie de recherche employée au travers de MAC par WoodStock est STR (*Simple Tabular Reduction*) (Ullmann, 2007) en associant à chaque contrainte, les quatre structures nécessaires à son implémentation. Enfin, l'heuristique *dom/ddeg* est utilisée afin de choisir les variables à résoudre par ordre croissant selon le ratio entre la taille du domaine courant et le degré dynamique des variables.

4.2.2. Phase de simulation : UCB

Rappelons dans un premier temps que tout programme GDL est un jeu séquentiel fini et que les utilités (les scores) de chaque joueur ne sont accessibles que dans un état terminal. De plus généralement, les instances de jeu utilisées en pratique impliquent un arbre de jeu possédant une profondeur et une largeur importante. MAC, à lui seul, ne peut donc pas résoudre l'ensemble de l'arbre de jeu en accord avec le temps délimité dans le cadre d'un match GGP.

C'est pourquoi, dans le but de choisir le plus judicieusement possible à chaque itération le prochain micro SCSP à résoudre, il est nécessaire de simuler les prochains états de l'arbre de jeu de tout état non-terminal afin d'estimer l'utilité des solutions qu'apporte MAC à chaque μSCSP_t. Dans ce but, nous utilisons une technique de bandits multi-bras dirigée par la propriété UCB (pour *Upper Confidence Bound*) (Browne *et al.*, 2012) en considérant chaque politique solution du μSCSP_t comme un « bras ». À partir d'une politique partielle du SCSP équivalent au jeu GDL associée à une politique solution du μSCSP_t, nous simulons uniformément et aléatoirement tous les coups possibles de $t+1$ à $T-1$.

Le principal problème est d'atteindre rapidement un état terminal sans résoudre chaque μSCSP rencontré. Heureusement, grâce à la propriété SAC que nous appliquons sur chaque μSCSP_t à chaque temps t, nous obtenons les coups légaux à chaque tour directement. Par conséquent, WoodStock peut choisir aléatoirement une sé-

quence d'actions jusqu'à atteindre un état terminal identifié par l'assignation de la valeur $true$ à la variable $terminal_t$.

La « meilleure » solution du μSCSP$_t$ est celle qui maximise $\bar{u}_i + \sqrt{\frac{2 \ln n}{n_i}}$, où $\bar{u}_i$ est le score moyen de la politique solution i, n_i est le nombre de fois où i a été simulé jusque là, et n est le nombre total de simulations réalisées. La résolution du prochain problème est décidée en instanciant μSCSP$_{t+1}$ par les valeurs de la meilleure politique solution estimée à partir de μSCSP$_t$.

Afin d'améliorer l'efficacité des techniques de filtrage de WoodStock une table de hachage de Zobrist (Zobrist, 1990) est utilisée pour stocker tous les états déjà explorés. En combinant les état terminaux déjà atteints avec le nombre de coups légaux de chaque état, il est possible de déterminer si un sous-arbre a été complètement exploré mais également d'éviter de simuler deux fois une même succession de coups légaux.

Finalement, le mouvement sélectionné par WoodStock correspond à l'action proposant la meilleure utilité espérée dans l'état courant. Notons que pour envisager tout problème de communication pouvant provoquer l'apparition d'un *timeout*, 2 secondes sont conservées pour sélectionner l'action et garantir son envoi au *game manager*.

L'algorithme 3 détaille le déroulement de MAC−UCB. Le *template* μSCSP$_t$ correspond au réseau de contraintes stochastiques à un niveau issue de la traduction du jeu GDL correspondant. Le second paramètre μSCSPs est une liste de réseaux stochastiques à un niveau (μSCSP$_i$) ordonnée de manière décroissante en fonction de l'utilité attendue (v_i) de chaque réseau. À l'état initial ($t = 0$), μSCSPs n'est composée que d'un seul réseau de contraintes stochastiques à un niveau correspondant au *template* μSCSP$_t$ et où les contraintes modélisant les règles « init » du jeu GDL correspondant sont ajoutées. Toutefois lors de l'appel de MAC−UCB à d'autres temps $t \neq 0$, μSCSPs ne contient que les réseaux de contraintes stochastiques à un niveau de temps supérieur à t associés à leurs utilités calculées précédemment.

MAC−UCB résout itérativement chaque μSCSP de la liste μSCSPs tant qu'il reste du temps (ligne 1). À chaque itération, le réseau $\mathcal{P}$ choisi pour être résolu est le premier élément de la liste μSCSPs correspondant au réseau associé à la plus grande utilité attendue et il est supprimé de la liste μSCSPs (ligne 2).

La résolution de $\mathcal{P}$ commence par la génération du CSP $\mathcal{P}'$ correspondant à la partie dure (ligne 3) et par la génération du μSCSP $\mathcal{P}''$ représentant la partie chance (ligne 4) à l'aide des algorithmes 1 et 2. $\mathcal{P}''$ est résolu par l'algorithme de *Forward Checking* adapté au cadre SCSP (ligne 5). Les lignes 6 à 13 illustrent la génération de la contrainte de faisabilité c_f à l'aide des solutions ($sols(\mathcal{P}'')$) du réseau $\mathcal{P}''$. c_f est ajoutée aux contraintes du réseau P' (ligne 14) et il est résolu par l'algorithme MAC (ligne 15).

Les lignes 16 à 25 permettent de générer et d'ajouter les réseaux $muSCSP_{t+1}$ à la liste μSCSPs afin de les résoudre aux prochaines itérations. La génération de chaque réseau au temps $t + 1$ est effectuée à partir de toute solution obtenue qui ne

correspond pas à un état terminal (ligne 17) détecté à l'aide de la variable $terminal$. Le réseau $\mu\texttt{SCSP}_i$ généré pour chaque solution τ_i est initialisé à l'aide du *template* $\mu\texttt{SCSP}_t$ (ligne 18). Puis pour chaque instanciation d'une variable f_{t+1} (correspondant au fluent au temps suivant $t+1$) du tuple solution τ_i, nous ajoutons une contrainte unaire restreignant le domaine de chaque variable f_t (correspondant au même fluent au temps t) du nouveau réseau $\mu\texttt{SCSP}_i$ au singleton $\{\tau_i[f_{t+1}]\}$ (lignes 19 et 20). L'utilité attendue v_i correspondant au réseau $\mu\texttt{SCSP}_i$ est générée à l'aide de la technique UCB (ligne 22). Finalement, le couple $(\mu\texttt{SCSP}_i, v_i)$ est inséré correctement dans la liste $\mu\texttt{SCSP}s$ en fonction de son utilité v_i (ligne 23).

Algorithme 3 : MAC−UCB

Données : Réseau *template* $\mu\texttt{SCSP}_t = (X_t, Y_t, D_t, C_t, P_t, 1)$, Liste ordonnée de réseaux stochastiques $\mu\texttt{SCSP}s = \{(\mu\texttt{SCSP}_0, v_0), \ldots, (\mu\texttt{SCSP}_n, v_n)\}$ avec $v_0 \geq \ldots \geq v_n$

1 **tant que** ¬ *timeout* **faire**
2 Sélectionner et supprimer le premier réseau $\mathcal{P}$ de $\mu\texttt{SCSP}s$
3 CSP $\mathcal{P}' = (X', D', C') \leftarrow partie\_dure(\mathcal{P})$
4 μSCSP $\mathcal{P}'' \leftarrow partie\_chance(\mathcal{P})$
5 $sols(\mathcal{P}'') \leftarrow \texttt{SFC}(\mathcal{P}'')$
6 Contrainte c_f = $(scp(c_f) \leftarrow \emptyset, rel(c_f) \leftarrow \emptyset)$
7 $scp(c_f) \leftarrow vars(\pi) \mid \pi \in sols(\mathcal{P}'')$
8 **pour chaque** $\pi_k \in sols(\mathcal{P}'')$ **faire**
9 **pour chaque** $(x = v) \in \pi_k \mid x \in V_t''$ **faire**
10 $\tau_k[x] \leftarrow v$
11 **fin**
12 $rel(c_f) \leftarrow rel(c_f) \cup \tau_k$
13 **fin**
14 $\mathcal{P}' = (X', D', C' \cup \{c_f\})$
15 $sols(\mathcal{P}') \leftarrow \texttt{MAC}(\mathcal{P}')$
16 **pour chaque** $\tau_i \in sols(\mathcal{P}')$ **faire**
17 **si** $\tau_i[terminal] = 0$ **alors**
18 $\mu\texttt{SCSP}_i = (X_i \leftarrow X_t, Y_i \leftarrow Y_t, D_i \leftarrow D_t, C_i \leftarrow C_t, P_i \leftarrow P_t, 1)$
19 **pour chaque** $\tau_i[f_{t+1}] \mid f_{t+1} \in X'$ **faire**
20 $C_i \leftarrow C_i \cup \{(f_t = \tau_i[f_{t+1}])\} \mid f_t \in X_i$
21 **fin**
22 $v_i \leftarrow \texttt{UCB}(\mu\texttt{SCSP}_i)$
23 $\mu\texttt{SCSP}s \leftarrow \mu\texttt{SCSP}s \cup \{(\mu\texttt{SCSP}_i, v_i)\}$ // dans l'ordre de $\mu\texttt{SCSP}s$
24 **fin**
25 **fin**
26 **fin**

Au cours de la section suivante, nous présentons les résultats de WoodStock au cours de sa première participation à une compétition GGP mais également au cours de la compétition en continue que propose le serveur *Tiltyard*.

5. Résultats compétitifs

Actuellement en compétition, `WoodStock` fonctionne sur un Intel Core i7-4770L CPU 3.50 Ghz associé à 32 Gb de RAM sous Linux. La première participation de `WoodStock` fut lors de la compétition `GGP` *Tiltyard Open* en décembre 2015 organisée par *Sam Schreiber* et *Alex Landau* sur le site Tiltyard [6].

5.1. Tiltyard Open 2015

5.1.1. Contexte

Cette compétition est réalisée en deux étapes. La première représente la phase de qualification où le temps de pré-traitement est fixé à 120 secondes et la temps de délibération à 15 secondes, la seconde étape représentant la finale propose un temps de pré-traitement de 180 secondes et le même temps de délibération.

Les participants de la phase de qualification jouent sur 10 nouveaux jeux au cours d'un tournoi suisse; la plupart d'entre eux sont réalisés plusieurs fois. Ils incluent des jeux à 1, 2, ou 4 joueurs et plusieurs d'entre eux sont simultanés.

Les programmes-joueurs participent automatiquement à chaque match sur la base des résultats de ces précédents matchs. Premièrement, les matchs sont réalisés entre joueurs possédant un score courant similaire sur le jeu courant et deuxièmement entre joueurs possédant un score courant similaire sur l'ensemble de la compétition. À cela s'ajoute un facteur permettant de réduire les matchs répétés entre les mêmes joueurs.

La table 5 indique les informations disponibles sur les 10 jeux proposés lors de la phase de qualification associée à un identifiant unique afin de conserver leurs anonymats. 1 à 4 joueurs sont impliqués dans le nombre de matchs prédéterminés pour chaque jeu. Chaque programme-joueur victorieux obtient un nombre de points égal au score qu'il a obtenu à l'issu du match pondéré par différents poids associés à chaque jeu.

Nous pouvons noter que la phase de qualification avantage les jeux à deux joueurs et que le score maximal pouvant être atteint est de $2,500$.

La finale inclut les quatre programmes-joueurs obtenant les meilleurs scores lors des qualifications et propose un système d'élimination sur plusieurs matchs réalisés par tour.

5.1.2. Résultats de `WoodStock` à l'issue de la compétition

Neuf programmes listés dans la table 6 ont participé à la *Tiltyard* Open 2015. Trois d'entre eux, fonctionnant avec un réseau de propositions (*propnet*) et des méthodes Monte Carlo, sont les derniers champions `GGP` de ces dernières années. La même

6. www.ggp.org/view/tiltyard/matches/\#tiltyard_open_20151204_2

Tableau 5. Les informations disponibles pour chaque jeu de la phase de qualification

Id	#Joueurs	#Matchs	Poids
1	1	1	2
2	2	6	0.5
3	1	2	1
4	2	6	0.5
5	1	1	2
6	2	6	0.5
7	4	4	0.5
8	2	3	1
9	4	4	0.5
10	2	3	1

table montre le score obtenu par chaque programme-joueur générique à l'issue de la phase de qualification.

Au cours de cette étape, `WoodStock` finit second avec un score (1366.250 points) plus élevé que les derniers champions `GGP`. Notons que `Galvanise` et `Sancho`, les deux derniers champions ne se sont pas qualifiés car ils ont perdu de nombreux points contre `WoodStock` et `LeJoueur`. `LeJoueur` (Méhat, Cazenave, 2011) est une implémentation en parallèle de l'ancien champion `GGP Ary` permettant de réaliser de nombreuses simulations. Ce dernier a obtenu la première place de la qualification en utilisant $3,000$ simulateurs et fonctionnant sur un cluster composé de 48 threads. Notons que pour le moment, `WoodStock` fonctionne sur un seul thread, par conséquent, les deux infrastructures ne sont pas similaires et explique pourquoi `LeJoueur` a obtenu un meilleur score.

Tableau 6. Les scores finaux suite à la phase de qualification

Rang	GGP player	Score final
1	LeJoueur	1783.750
2	WoodStock	1366.250
3	GreenShell [7]	1351.000
4	QFWFQ	1245.875
5	Sancho	1213.500
6	SteadyEddie	1201.875
7	Galvanise	1185.250
8	Modest	1129.000
9	MonkSaki	1037.813

Pour sa première participation, WoodStock s'est qualifié pour la finale au côté de LeJoueur, GreenShell et QFWFQ. Malheureusement, au cours de cette phase, WoodStock a rencontré une erreur sur le premier jeu GDL et il était dans l'impossibilité d'envoyer une action au *game manager*. Cette erreur était due à un problème dans son implémentation. Par conséquent, ces mouvements ont été remplacés par des actions aléatoires et il a perdu les deux premiers matchs. Sur les trois matchs suivants, WoodStock n'a rencontré aucun problème et il les a remportés pour finalement obtenir la troisième position de la compétition GGP. Le tableau 7 indique les rangs finaux.

Tableau 7. Le classement final de la Tiltyard Open 2015

Rang	GGP player
1	LeJoueur
2	GreenShell
3	WoodStock
4	QFWFQ

5.2. Tournoi continu GGP

Tableau 8. Le classement au 15 juillet 2016 du tournoi continu GGP

Rang	GGP player	Score Agon
1	WoodStock	225.4
2	Sancho	220.63
3	Galvanise	214.4
4	LabThree	191.71
5	Coursera Quixote	169.77
6	SgianDubh	163.92
7	CloudKingdom	162.23
8	SteadyEddie	159.53
9	Alloy	133.5
10	LeJoueur	120.26

Suite à sa première participation en compétition GGP, l'erreur d'implémentation rencontrée par WoodStock a été corrigée et il a pu participer à la compétition continue de *General Game Playing* où l'ensemble des programmes-joueurs sont représentés (à ce jour, plus de $1,000$ programmes) jouant sur l'ensemble des jeux que propose le serveur Tiltyard. Ce type de compétition est naturellement plus complexe car le temps de pré-traitement et le temps de délibération sont choisis aléatoirement entre

7. Il est nécessaire de mentionner que GreenShell est le nouveau pseudonyme utilisé par TurboTurtle, le champion 2011 et 2013 de GGP.

deux matchs et aucun programme-joueur ne peut donc se calibrer en fonction de ce paramètre.

Depuis mars 2016 et après près de $2,000$ matchs, `WoodStock` est le leader du tournoi et a détrôné `Sancho` qui était leader de cette compétition depuis plus de trois ans. À ce jour, `WoodStock` obtient un score moyen supérieur à 76 et le tableau 8 indique le classement du 15 juillet 2016 du tournoi continu `GGP`.

Le classement actuel et les matchs de `WoodStock` sont accessibles en direct à l'adresse suivante : www.ggp.org/view/tiltyard/players/WoodStock. Il est notamment possible de retrouver le score moyen obtenu par `WoodStock` sur chaque jeu `GDL`.

6. Optimisation

Conceptuellement, tout jeu (déterministe ou stochastique) à horizon fini impliquant deux joueurs et à somme nulle possède une fonction optimale spécifiant le résultat attendu d'un jeu, pour tout état possible, sous la condition que l'ensemble des joueurs jouent parfaitement.

En théorie de tels jeux peuvent être résolus en calculant récursivement la fonction d'utilité dans un arbre de recherche de taille l^d, où l est la largeur du jeu (le nombre d'actions légales par état) et d est la profondeur du jeu (le nombre de mouvements nécessaires pour atteindre un état terminal). Pourtant, même pour des jeux d'une taille modérée, une recherche exhaustive n'est pas envisageable en pratique lors des tournois de *General Game Playing*, dû au temps de délibération très court imposé à chaque joueur pour déterminer sa prochaine action.

C'est pour cette raison, que l'un des challenges de `GGP` est de concevoir des techniques génériques d'inférence et d'apprentissage pour réduire efficacement l'espace de recherche des jeux, dont les règles ne sont fournies qu'au début du jeu.

À cette fin, la détection de symétries est une approche d'inférence bien connue en Intelligence Artificielle pour accroître la résolution de tels problèmes combinatoires, en transférant les connaissances apprises en des régions équivalentes de l'espace de recherche. Les jeux représentent un exemple notable car ils impliquent typiquement de nombreux états équivalents et des actions équivalentes. De telles similarités peuvent être exploitées pour transférer la fonction d'utilité entre plusieurs nœuds de l'arbre de recherche : la largeur l de l'arbre peut être réduite en exploitant les actions menant à des résultats attendus similaires et la profondeur d de l'arbre peut également être réduite en reconnaissant des états associés à des valeurs similaires.

Ainsi la plupart des méthodes de détections de symétries sont réalisées en utilisant des algorithmes basés sur des automorphismes de graphes (Darga *et al.*, 2008 ; McKay, Piperno, 2014). Le composant principal pour mettre en évidence les symétries de jeux est une structure graphique sur laquelle un groupe de permutation est induit.

Rappelons qu'une symétrie dans un ensemble D est une permutation sur D ou, de manière équivalente une bijection σ de D vers D. De nombreuses types de symétries ont été proposées dans la littérature de la programmation par contraintes, notamment les symétries de solutions qui préservent l'ensemble des solutions et les symétries de contraintes qui préservent l'ensemble des contraintes. (Cohen *et al.*, 2006) montre que chaque symétrie de contrainte correspond à un automorphisme de la micro-structure complémentaire du réseau de contraintes et que chaque symétrie de solution correspond à un automorphisme de la micro-structure complémentaire enrichie par l'ensemble des instanciations globalement incohérentes.

Afin de détecter les symétries, `WoodStock` doit générer la micro-structure complémentaire $\mathcal{H}_t$ de chaque $\mu\texttt{SCSP}_t$. Pour cela, il commence par générer la micro-structure complémentaire commune à chaque $\mu\texttt{SCSP}_t$ issue de la traduction du programme `GDL` à tout temps t (sans les règles *init*). De ce fait pour obtenir $\mathcal{H}_t$ à un temps t donné, il suffit pour le temps $t = 0$ d'ajouter les arêtes correspondantes aux contraintes issues des règles *init* et pour $t \neq 0$ les arêtes correspondantes aux contraintes générées à partir des solutions de $\mu\texttt{SCSP}_{t-1}$. Les micro-structures obtenues permettent de capturer graphiquement les règles d'un jeu `GDL` à chaque temps t. `WoodStock` représente chaque micro-structure complémentaire par un hypergraphe pouvant aussi être considéré comme un graphe bipartie : La première partie représentant l'ensemble des littéraux (couples variables-valeurs) et la seconde partie modélisant l'ensemble des tuples interdits composants toutes les contraintes de $\mathcal{P}$. Une arête (l, τ) de ce graphe correspond à l'occurrence d'un littéral l dans un tuple τ d'une contrainte.

Notons que pour chaque $\mu\texttt{SCSP}_t$, l'ensemble des contraintes est enrichi de nouvelles contraintes modélisant les stratégies. Ainsi, chaque $\mu\texttt{SCSP}_t$ $\mathcal{P}$ est enrichi d'une contrainte de portée $(\{F_t\}, \{A_t\}, A_{k+1,t})$ (où F_t est l'ensemble des fluents à l'instant t, A_t l'ensemble des actions des k joueurs et $A_{k+1,t}$ les actions du joueur environnement $k+1$) et de relation formée par l'ensemble des tuples de la forme $(s, \boldsymbol{a})$, s étant défini sur $\{F_t\}$ et $\boldsymbol{a}$ représentant une combinaisons d'actions sur $\{A_t\}$ et $A_{k+1,t}$. Chaque tuple $(s, \boldsymbol{a})$ est vu comme une instanciation globalement incohérente si toutes les politiques débutant à l'état s avec le vecteur d'action $\boldsymbol{a}$ est prédite par `UCB` comme une politique non solution. Dit autrement, l'utilité attendue simulée par `UCB` est inférieure au seuil θ.

Sur la base de chaque $\mu\texttt{SCSP}$ $\mathcal{P}$ enrichi de toute instanciation globalement incohérente détectée au fil de la résolution du problème, `WoodStock` déduit les automorphismes de la micro-structure complémentaire de $\mathcal{P}$ à l'aide de `NAUTY` (McKay, Piperno, 2014) et génère les $\mu\texttt{SCSP}$s symétriques. Chacun de ces $\mu\texttt{SCSP}$ est stocké et associé à leur utilité attendue dans une table de hachage de *Zobrist* (Zobrist, 1990) où chaque valeur assignée dans chaque $\mu\texttt{SCSP}$ correspond à une valeur de hachage générée aléatoirement. Le nombre de *Zobrist* alors utilisé pour retrouver un $\mu\texttt{SCSP}$ dans cette table est alors le `XOR` des nombres aléatoires associés à chaque couple variable-valeur assignée garantissant une probabilité de collision très faible. Ainsi avant de résoudre un $\mu\texttt{SCSP}$, `WoodStock` vérifie si il ne correspond pas à un $\mu\texttt{SCSP}$ symé-

trique déjà rencontré. Si c'est le cas, l'utilité attendue associée est directement obtenue sans avoir besoin de le résoudre ou de simuler les état suivants permettant ainsi un gain de temps substantiel.

6.1. WoodStock, Champion GGP 2016

Lors de la dernière compétition internationale organisée par l'université de *Stanford* (IGGPC'16), nous avons pu mettre en pratique WoodStock optimisé par la détection de symétries. La compétition a regroupé 10 concurrents comprenant les champions GGP 2011, 2013, 2014 et 2015 : Alloy, DROP-TABLE-TEAMS, Galvanise, General, MaastPlayer, QFWFQ, Sancho, SteadyEddie, TurboTurtle et WoodStock. Ils se sont affrontés au travers de nombreux jeux à un ou deux joueurs au cours de deux journées. Tous les jeux à deux joueurs étaient garantis à somme nulle. La plupart des jeux GDL étaient nouveaux et mis en pratique pour la première fois en compétition. 120 secondes de temps de pré-traitement (*StartClock*) et 30 secondes de temps de délibération (*PlayClock*) ont été allouées.

6.1.1. Jour 1

Lors de la première journée, deux groupes regroupant chacun 5 joueurs ont été réalisés arbitrairement. Le classement final de chaque groupe est obtenu à la suite du calcul de la somme pondérée des scores de chaque joueur. Les deux joueurs obtenant les scores maximaux sont qualifiés au sein de la « poule des vainqueurs » et les deux suivants au sein de la « poule des vaincus » pour la seconde journée. Les scores obtenus par WoodStock sur chaque jeu à l'issue de la première journée sont présentés dans le tableau 9. WoodStock a obtenu 1.200 points suite à cette première journée et il fut le seul programme-joueur à obtenir le score maximum sur l'ensemble des puzzles. Ainsi, il fut qualifié au sein de la « poule des vainqueurs » pour la seconde journée au coté de DROP-TABLE-TEAMS, Galvanise et TurboTurtle.

6.1.2. Jour 2

La seconde journée s'est déroulée sous la forme de duels remportés par le premier programme obtenant 3 victoires (excepté pour la finale où 4 victoires sont nécessaires). Le tableau 10 répertorie l'ensemble des matchs joués par WoodStock. Au cours des quarts de finale, WoodStock a affronté DROP-TABLE-TEAMS et a obtenu 3 victoires et 2 défaites lui permettant d'atteindre les demi-finales face à TurboTurtle. Au cours de ces dernières, WoodStock n'a rencontré aucune défaite et a obtenu 3 victoires. Finalement, lors de la finale l'opposant à Galvanise, WoodStock est devenu Champion GGP à la suite de 2 défaites et de 4 victoires.

Un point important à noter est que la détection de symétries a été primordiale au cours de la compétition, en particulier lors du dernier jeu pratiqué par WoodStock. Lors des deux matchs de Jointconnectfour, un grand nombre de symétries détectées ont permis de réduire fortement l'espace de recherche et d'explorer complète-

ment l'arbre de recherche au bout d'une trentaine de coups joués contrairement à son opposant.

Tableau 9. `WoodStock`*- IGGPC 2016 - Jour 1.*

Jeu	#Joueurs	WoodStock	Poids
Vectorracer6	1	100	1
EightPuzzle	1	100	1
HunterBig	1	100	1
UntwistyComplex	1	100	0.5
Jointbuttonsandlights25	1	100	0.5
Sudoku	1	100	0.5
Bandl3	1	100	0.33
bandl7	1	100	0.33
bandl10	1	100	0.33
Majorities	2	100 / 300	1
Battleofnumbers	2	150 / 300	1
Jointconnectfour77	2	100 / 300	1
Breakthrough	2	300 / 300	1

Tableau 10. `WoodStock`*- IGGPC 2016 - Jour 2.*

Jeu	WoodStock	Adversaire
Quart de finale contre `DROP-TABLE-TEAMS`		
Hexawesome	100	0
Reflectconnect6	30	70
Connectfour77	100	0
Connectfour77	0	100
Majorities	100	0
Demi finale contre `TurboTurtle`		
Battleofnumbersbig	100	0
Platformjumpers	100	0
Jointconnectfour77	100	0
Finale contre `Galvanise`		
Reversi	0	100
Reversi	0	100
Skirmish	100	0
Skirmish	100	0
Jointconnectfour	100	0
Jointconnectfour	100	0

7. Conclusion

Dans cet article, nous décrivons un nouveau type de programme-joueur générique basé sur la programmation par contraintes stochastiques illustrée par WoodStock au travers d'un programme orienté objet. Pour réaliser la communication entre un programme-joueur GGP et le serveur de jeux, nous fournissons un joueur réseau GGP dénommé spy-ggp adaptable à différents langages de programmation. WoodStock permet de mettre en pratique MAC-UCB dans le contexte d'une compétition GGP, tout d'abord en implémentant le processus de traduction d'un programme GDL vers un SCSP puis en réalisation les phases de résolution et de simulation nécessaires.

À l'aide de cet algorithme, notre programme-joueur mono-thread, WoodStock est actuellement leader de la compétition continue sur le serveur *Tiltyard* et suite à la compétition GGP dénommée *Open Tiltyard* 2015, il s'est classé second lors des qualifications avec un score plus élevé que trois des anciens champions GGP et troisième en finale derrière LeJoueur, un programme-joueur multi-threads.

Par la suite, nous avons optimisé WoodStock en ajoutant à son implémentation la détection de symétries issue de la programmation par contraintes et permettant de réduire l'espace de recherche en évitant d'explorer des actions et des états dont l'utilité attendue est directement déductible des symétries détectées. Ainsi, avec cette optimisation, WoodStock a participé pour la première fois à la compétition internationale de *General Game Playing* 2016 organisée par l'université de *Stanford*. À la suite de celle-ci, WoodStock a remporté la compétition et il est devenu le champion GGP 2016.

Améliorations et perspectives

L'objectif de ce papier est de confirmer l'efficacité d'un programme-joueur générique dirigé par les contraintes. Cependant, WoodStock implémente ce formalisme au travers d'un programme orienté objet permettant de l'améliorer continuellement en ajoutant différents modules au programme.

Par exemple, WoodStock utilise STR pour réaliser la stratégie de recherche de l'algorithme mais (Lecoutre *et al.*, 2015) fournit un nouvel algorithme beaucoup plus performant pour la résolution d'instances de grande taille, pouvant ainsi être intéressant à implémenter dans le cadre des compétitions GGP. De même, l'utilisation d'UCB pour réaliser les simulations pourrait être remplacée par une technique de bandits plus adaptée au cadre GGP. Une autre voie naturelle serait de paralléliser les phases de résolution et de simulation afin d'être compétitif avec les programmes-joueurs multi-threads.

Finalement, le modèle stochastique utilisé pourrait être étendu au cadre des jeux à information incomplète décrit en GDL-II en intégrant la traduction du mot-clé *sees* permettant de décrire les observations de chaque joueur en fonction de l'état courant. Une étude en ce sens est en cours ainsi que l'implémentation d'un algorithme adapté à cette classe de jeux.

Bibliographie

Browne C., Powley E., Tavener S. (2012). A survey of monte carlo tree search methods. *Intelligence and AI*, vol. 4, n° 1, p. 1–49.

Cazenave T., Mehat J. (2010). Ary - a general game playing program. In *Proc. of board games studies colloquium.*

Clune J. (2007). Heuristic evaluation functions for general game playing. In *Proc. of AAAI'07*, p. 1134–1139.

Cohen D. A., Jeavons P., Jefferson C., Petrie K. E., Smith B. M. (2006). Symmetry definitions for constraint satisfaction problems. *Constraints*, vol. 11, n° 2-3, p. 115–137.

Cox E., Schkufza E., Madsen R., Genesereth M. (2009). Factoring general games using propositional automata. In *Ijcai'09 workshop on general game playing (giga'09)*, p. 13-20.

Darga P. T., Sakallah K. A., Markov I. L. (2008). Faster symmetry discovery using sparsity of symmetries. In *Dac'08*, p. 149–154. ACM.

Debruyne R., Bessière C. (1997). Some practicable filtering techniques for the constraint satisfaction problem. In *Ijcai'97*, p. 412–417.

Draper S., Rose A. (2014). *Sancho.* http://sanchoggp.blogspot.fr/2014/07/sancho-is-ggp-champion-2014.html.

Emslie R. (2015). *Galvanise.* https://bitbucket.org/rxe/galvanise_v2.

Finnsson H., Björnsson Y. (2008). Simulation-based approach to General Game Playing. In *Aaai'08*, p. 259–264. AAAI Press.

Finnsson H., Björnsson Y. (2011). Cadiaplayer: Search-control techniques. *KI - Künstliche Intelligenz*, vol. 25, n° 1, p. 9–16.

Genesereth M., Björnsson Y. (2013). The international general game playing competition. *AI Magazine*, vol. 34, n° 2, p. 107-111.

Genesereth M., Love N. (2005). General game playing: Overview of the aaai competition. *AI Magazine*, vol. 26, p. 62–72.

Koriche F., Lagrue S., Piette É., Tabary S. (2015). Résolution de scsp avec borne de confiance pour les jeux de stratégie. In *Jfpc'15*, p. 160-169.

Koriche F., Lagrue S., Piette É., Tabary S. (2016). General game playing with stochastic csp. *Constraints*, vol. 21, n° 1, p. 95–114.

Lecoutre C., Likitvivatanavong C., Yap R. H. C. (2015). STR3: A path-optimal filtering algorithm for table constraints. *Artificial Intelligence*, vol. 220, p. 1–27.

Lifschitz V., Yang F. (2011). Eliminating function symbols from a nonmonotonic causal theory. In *Knowing, reasoning, and acting: Essays in honour of hector j. levesque.* College Publicaions.

Littman M. L., Majercik S. M., Pitassi T. (2001). Stochastic boolean satisfiability. *Journal of Automated Reasoning*, vol. 27, n° 3, p. 251–296.

Love N., Genesereth M., Hinrichs T. (2006). *General game playing: Game description language specification.* Rapport technique n° LG-2006-01. Stanford University.

McKay B. D., Piperno A. (2014). Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, vol. 60, n° 0, p. 94 - 112.

Mehat J., Cazenave T. (2008). *An account of a participation to the 2007 general game playing competition.* http://www.ai.univ-paris8.fr/~jm/ggp/ggp2008-2.pdf. (unpublished)

Möller M., Schneider M. T., Wegner M., Schaub T. (2011). Centurio, a general game player: Parallel, Java- and ASP-based. *Künstliche Intelligenz*, vol. 25, n° 1, p. 17-24.

Méhat J., Cazenave T. (2011). A parallel general game player. *KI - Künstliche Intelligenz*, vol. 25, n° 1, p. 43-47.

Sabin D., Freuder E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. In *ECAI'94*, p. 125–129.

Schiffel S., Thielscher M. (2007). Fluxplayer: A successful general game player. In *Aaai'07*, p. 1191–1196. AAAI Press.

Schreiber S. (2014). *Ggp.org - tiltyard gaming server.* http://tiltyard.ggp.org/. (CS227b class at Stanford University)

Silver D., Huang A., Maddison C. J., Guez A., Sifre L., Driessche D. v. d. *et al.* (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, vol. 529, p. 484–503.

Sturtevant N. R. (2008). An analysis of UCT in multi-player games. *International Computer Games Association Journal*, vol. 31, n° 4, p. 195–208.

Thielscher M. (2005). Flux: A logic programming method for reasoning agents. *Theory Pract. Log. Program.*, vol. 5, n° 4-5, p. 533-565.

Thielscher M. (2010). A general game description language for incomplete information games. In *Aaai'10.*

Thielscher M. (2011a). GDL-II. *KI - Künstliche Intelligenz*, vol. 25, n° 1, p. 63–66.

Thielscher M. (2011b). The general game playing description language is universal. In *Ijcai'11*, p. 1107–1112. AAAI Press.

Ullmann J. R. (2007). Partition search for non-binary constraint satisfaction. *Information Sciences*, vol. 177, n° 18, p. 3639–3678.

Walsh T. (2009). Stochastic constraint programming. *Computing Research Repository*, vol. abs/0903.1152.

Zobrist A. (1990). A new hashing method with application for game playing. *International Computer Games Association Journal*, vol. 13, n° 2, p. 69–73.

Le problème de satisfaction de contraintes quantifiées et les jeux à deux joueurs à horizon fini : le projet QuaCode

Vincent Barichard, Igor Stéphan

Université d'Angers, 2 boulevard Lavoisier, 49045 Angers, France
vincent.barichard@univ-angers.fr,igor.stephan@univ-angers.fr

RÉSUMÉ. *Le problème de satisfaction de contraintes quantifiées (QCSP) est une généralisation du problème de satisfaction de contraintes (CSP) pour lequel les variables peuvent être quantifiées aussi bien existentiellement qu'universellement. Le concept de QCSP offre un cadre naturel pour traiter des problèmes PSPACE comme par exemple les jeux à deux joueurs à horizon fini et information complète ou la planification sous incertitude. Nous présentons à travers trois exemples comment les QCSP peuvent être utilisés pour modéliser des jeux à deux joueurs : le* Jeu de Nim*, le* MatrixGame *et le* Puissance Quatre*. Les solveurs QCSP de l'état de l'art ont un défaut majeur : ils explorent un espace combinatoire plus important que l'espace de recherche naturel du problème original car ils sont inaptes à reconnaître que certains sous-problèmes sont nécessairement vrais. Nous proposons dans le cadre des jeux à deux joueurs à horizon fini une solution efficace pour utiliser les solveurs QCSP et une réponse aussi simple qu'élégante à ce parcours superflu. Nous proposons un solveur QCSP construit au-dessus de la librairie CSP GeCode que nous comparons aux solveurs de l'état de l'art.*

ABSTRACT. *Quantified Constraint Satisfaction Problems (QCSP) are a generalization of Constraint Satisfaction Problems (CSP) in which variables may be quantified existentially and universally. QCSP offers a natural framework to express PSPACE problems as finite two-player games with perfect information or planing under uncertainty. We present how QCSP may be used to model two-player games on three classical games : Nim game, MatrixGame and Connect Four. State-of-the-art QCSP solvers have an important drawback: they explore much larger combinatorial space than the natural search space of the original problem since they are unable to recognize that some sub-problems are necessarily true. We propose a very simple and elegant solution to use efficiently QCSP to design finite two-player games. Our QCSP solver built over GeCode, a CSP library, obtained very good results compared to state-of-the-art QCSP solvers.*

MOTS-CLÉS : *jeu à deux joueurs à horizon fini, problème de satisfaction de contraintes quantifiées, QCSP.*

KEYWORDS: *finite two-players game, quantified constraint satisfaction problem, QCSP.*

DOI:10.3166/RIA.31.337-365

1. Introduction

Le problème de satisfaction de contraintes quantifiées (ou QCSP pour *Quantified Constraint Satisfaction Problem*) est une généralisation du problème de satisfaction de contraintes (ou CSP pour *Constraint Satisfaction Problem*) dans laquelle les variables peuvent être non seulement quantifiées existentiellement (comme dans les CSP) mais aussi universellement (Bordeaux, Monfroy, 2002). Un QCSP est une alternance de variables existentiellement ou universellement quantifiées portant sur des domaines finis suivie par un CSP. Un CSP est alors une collection de contraintes portant sur des variables prenant leurs valeurs dans des domaines finis. Un QCSP peut être vu comme un jeu à deux joueurs à horizon fini dans lequel les variables existentiellement quantifiées représentent les coups d'un joueur A et les variables universellement quantifiées les coups d'un joueur B. Un QCSP est valide si le joueur A a une stratégie pour gagner c'est-à-dire une stratégie pour affecter des valeurs aux variables existentiellement quantifiées de telle manière que, quelles que soient les valeurs affectées aux variables universelles par le joueur B, le CSP est vrai. Le joueur A tente ainsi de rendre vraies toutes les contraintes du CSP tandis que le joueur B tente au moins d'en violer une.

Voici un exemple issu de la combinatoire : le problème du boulanger. Ce problème consiste à aider un boulanger souhaitant acquérir quatre poids distincts à choisir dans l'intervalle $\{1, \ldots, 40\}$ kg, lui permettant de peser n'importe quelle quantité entière de farine dans l'intervalle $\{1, \ldots, 40\}$ kg en utilisant une balance de Roberval. Pour la pesée, chaque poids p_i peut être placé de n'importe quel côté de la balance ou ne pas être utilisé. Ce problème peut être modélisé par le QCSP suivant : sachant que $p_1, p_2, p_3, p_4, f \in \{1, \ldots, 40\}$, $e_1, e_2, e_3, e_4 \in \{-1, 0, 1\}$, alors

$$\exists p_1 \exists p_2 \exists p_3 \exists p_4 \forall f \exists e_1 \exists e_2 \exists e_3 \exists e_4 (\sum_{i=1}^{4} p_i \times e_i = f).$$

Les emplacements e_i indiquent pour chaque poids p_i s'il est placé sur le plateau à côté de la farine (-1), sur l'autre plateau (1) ou absent de la balance (0). Pour chaque poids de farine à peser les emplacements sont différents et dépendent de ce poids. Le problème du boulanger admet une unique stratégie gagnante $\{1, 3, 9, 27\}$ (modulo les permutations des variables p_i) si on considère que ce qui importe au boulanger est de savoir quels poids il doit acheter. Sinon le reste de la stratégie gagnante est une fonction $\{1, \ldots, 40\} \times \{1, 2, 3, 4\} \to \{-1, 0, 1\}$.

L'étude des QCSP est récente mais il existe un réel intérêt pour mettre au point des techniques efficaces pour résoudre de tels problèmes (Chen *et al.*, 2015 ; Börner *et al.*, 2009 ; Gent *et al.*, 2004 ; 2008 ; 2005 ; Bordeaux *et al.*, 2005 ; Benedetti *et al.*, 2007 ; Bordeaux, Zhang, 2007 ; Börner *et al.*, 2003 ; Benedetti *et al.*, 2008 ; Verger, Bessiere, 2008 ; Pralet, Verfaillie, 2011 ; Mamoulis, Stergiou, 2004). Cette extension est pleine de promesses car elle permet de coder de manière plus compacte certains problèmes et même d'en modéliser d'autres qui ne peuvent l'être en CSP. Mais cette extension accroît la complexité du problème de décision, passant de NP-complet (pour un CSP classique) à PSPACE-complet. De même, le calcul d'au moins une stratégie gagnante

pour un QCSP relève de la classe de complexité PSPACE (Stockmeyer, Meyer, 1973). Il n'y a donc qu'un mince espoir (gardant les hypothèses classiques de la théorie de la complexité (Papadimitriou, 1994)) d'obtenir un algorithme efficace pour résoudre un QCSP quelconque. Mais il en est de même pour SAT et CSP dont la complexité est NP-complet or nous connaissons les succès des solveurs SAT et CSP pour certaines classes de problèmes. Il est possible qu'il en soit de même pour PSPACE, comme le suggère les nombreuses classes non triviales de QCSP, dans le cadre des jeux combinatoires, identifiées comme étant polynomiale en temps (Börner *et al*., 2009).

Le domaine des QCSP est à l'intersection de deux domaines : le domaine des CSP (Tsang, 1993) et celui des QBF (Cadoli *et al*., 1998) (les formules booléennes quantifiées ou QBF étant des QCSP restreintes aux booléens). Il n'y a, à l'aune de nos connaissances que trois solveurs QCSP, excepté le nôtre, QuaCode (Barichard, Stéphan, 2014 ; Barichard, Stéphan, 2014) : BlockSolve (Verger, Bessiere, 2006), Queso (Gent *et al*., 2008) et Qecode (Benedetti *et al*., 2008). BlockSolve est basé sur un algorithme de Fourier-Motzkin par élimination de quantificateurs. Queso est basé sur un algorithme de recherche quantifié et son approche est très proche de la nôtre. Qecode est aussi un solveur basé sur un algorithme de recherche quantifié mais est dédié à une extension des QCSP (QCSP+), qui propose en plus une forme restreinte de la quantification, et qui est utilisé principalement pour spécifier des jeux finis à deux joueurs. Ce dernier solveur est toujours maintenu tandis que les deux autres ne le sont plus.

Comme un solveur QCSP basé sur un algorithme de recherche quantifié peut être vu comme une extension d'un solveur CSP basé sur un algorithme de recherche, il est particulièrement pertinent de chercher à construire un solveur QCSP au-dessus de technologies pour solveur CSP sans les changer. Cette approche est la nôtre : nous implantons QuaCode, un solveur QCSP, au-dessus de GeCode, une bibliothèque de classes pour la gestion des contraintes d'un CSP, sans changer le cœur de la bibliothèque.

La plupart des solveurs de l'état de l'art, tout comme QuaCode ont un inconvénient majeur : ils parcourent un espace combinatoire bien plus grand que l'espace de recherche du problème original. Si, pour la plupart des problèmes, il n'y a pas de solution systématique, il n'en va pas de même pour les jeux à deux joueurs à horizon fini. Nous proposons dans ce cadre un théorème qui prouve l'élimination du parcours superflu.

Nous présentons tout d'abord en section 2 le concept de problème de satisfaction de contraintes quantifiées. Nous spécifions, en section 3, trois jeux à deux joueurs à horizon fini en QCSP : le *Jeu de Nim*, le *MatrixGame* ainsi que le *Puissance Quatre*. Nous montrons en particulier toute l'élégance de ces spécifications en QCSP. Nous abordons, en section 4, la spécification d'un solveur QCSP comme extension d'un solveur CSP. Nous présentons, en section 5, le « talon d'Achille » des QCSP, c-à-d. le problème de l'exploration des espaces de recherche inutile par construction dans les solveurs QCSP comme extensions de solveurs CSP et la solution que nous y apportons. Enfin, en section 6, nous présentons notre solveur QCSP QuaCode qui intègre

notre solution au problème du « talon d'Achille » et le comparons à l'état de l'art. Nous continuons par une discussion dans laquelle nous présentons QuaCode comme un système ouvert, capable en tant que solveur complet de coopérer avec des méthodes incomplètes via une interface pour orienter sa recherche. Nous concluons en donnant un ensemble de perspectives.

2. Problème de satisfaction de contraintes quantifiées

2.1. Syntaxe des QCSP

Le symbole $\mathcal{PS}$ représente les symboles propositionnels, le symbole $\exists$ représente le quantificateur existentiel et le symbole $\forall$ représente le quantificateur universel. Le symbole $\wedge$ représente la conjonction logique, le symbole $\vee$ représente la disjonction logique, le symbole $\rightarrow$ représente l'implication logique, le symbole $\leftrightarrow$ représente l'équivalence logique, le symbole $\top$ représente ce qui est toujours vrai et le symbole $\bot$ représente ce qui est toujours faux. Le symbole $\equiv$ représente l'équivalence entre formules.

Un QCSP est un quintuplet $(\mathbf{V}, \mathbf{order}, \mathbf{quant}, \mathbf{D}, \mathbf{C})$: $\mathbf{V}$ est un ensemble de n variables, **order** est une bijection de $\mathbf{V}$ dans $[1..n]$, **quant** est une fonction de $\mathbf{V}$ dans $\{\exists, \forall\}$ ($\mathbf{quant}(v)$ dénote le quantificateur associé à la variable v), $\mathbf{D}$ est une fonction de $\mathbf{V}$ dans l'ensemble des domaines $\{D(v_1), \ldots, D(v_n)\}$ telle que, pour toute variable $v_i \in \mathbf{V}$, $D(v_i)$ en dénote son domaine, i.e. l'ensemble fini de toutes les valeurs possibles ($D(v)$ est le domaine associé à la variable v), $\mathbf{C}$ est un ensemble de contraintes. Si $v_{j_1}, \ldots, v_{j_m}$ sont les variables d'une contrainte $c_j \in \mathbf{C}$ alors la relation associée à c_j est un sous-ensemble du produit cartésien $D(v_{j_1}) \times \cdots \times D(v_{j_m})$. Dans ce qui suit, pour chaque $i \in [1..n]$, $q_{v_i} = \mathbf{quant}(v_i)$ et $D_{v_i} = D(v_i)$.

Un QCSP $(\mathbf{V}, \mathbf{order}, \mathbf{quant}, \mathbf{D}, \mathbf{C})$ est généralement représenté par sa formule en logique du premier ordre $q_{v_1} v_1 \ldots q_{v_n} v_n \bigwedge_{c_j \in \mathbf{C}} c_j$ avec $v_1 \in D_{v_1}$, …, $v_n \in D_{v_n}$, $\mathbf{order}(v_i) = i$, pour chaque $i \in [1..n]$. Avec cette représentation $q_{v_1} v_1 \ldots q_{v_n} v_n$ est appelé « lieur » (un lieur vide étant noté ε).

Par exemple, le QCSP $(\{x, y, z, t\}, \mathbf{order}, \mathbf{quant}, \{\{0, 1, 2\}\}, \mathbf{C})$ avec

$$\begin{cases} \mathbf{order} = \{(1, x), (2, y), (3, z), (4, t)\}, \\ \mathbf{quant} = \{(x, \exists), (y, \exists), (z, \forall), (t, \exists)\}, \\ D(x) = D(y) = D(z) = D(t) = \{0, 1, 2\}, \\ \mathbf{C} = \{(x = (y * z) + t), (t \leq x)\} \end{cases}$$

est noté : $\exists x \exists y \forall z \exists t((x = (y * z) + t) \wedge (t \leq x))$ avec $x, y, z, t \in \{0, 1, 2\}$.

2.2. Sémantique des QCSP

Une stratégie est un arbre pour un lieur $Q = q_{v_1} v_1 \ldots q_{v_n} v_n$ avec $v_1 \in D_{v_1}$, …, $v_n \in D_{v_n}$, $Q = q_{v_1} v_1 \ldots q_{v_n} v_n$ est un arbre tel que :

– chaque nœud feuille est étiqueté avec le symbole $\square$ et à la profondeur n,

– chaque nœud interne à la profondeur k, $0 \leq k < n$, est étiqueté avec la variable v_{k+1},

– chaque arc reliant un nœud de profondeur k à l'un de ses fils est étiqueté par un élément de D_{v_k},

– toutes les étiquettes des arcs reliant un nœud à l'un de ses fils sont différentes,

– chaque nœud étiqueté par une variable existentiellement quantifiée admet un unique fils et

– chaque nœud étiqueté par une variable universellement quantifiée dont le domaine est de taille k admet k nœuds fils.

Un scénario est une séquence d'étiquettes $val_1, \ldots, val_n$ sur un chemin (v_1, val_1), …, (v_n, val_n), $val_i \in D_{v_i}$ pour tout i, $1 \leq i \leq n$, d'une stratégie pour un lieur $q_{v_1} v_1 \ldots q_{v_n} v_n$. Un scénario correspond à la notion d'instanciation dans un CSP où toutes les variables sont instanciées à une valeur de leur domaine. Un scénario val_1, …, val_n pour un QCSP $(\mathbf{V}, \mathbf{order}, \mathbf{quant}, \mathbf{D}, \mathbf{C})$ tel que $\mathbf{V} = \{v_1, \ldots, v_n\}$ est un scénario prometteur si $(\bigwedge_{1 \leq i \leq n} v_i = val_i) \wedge (\bigwedge_{c_j \in \mathbf{C}} c_j)$ est vrai ; un tel scénario correspond à l'instanciation complète $v_1 = val_1$, …, $v_n = val_n$; c'est un scénario prometteur car il satisfait toutes les contraintes. Un scénario prometteur correspond à la notion de solution dans un CSP. Une stratégie est une stratégie gagnante si tous les scénarios qui la composent sont des scénarios prometteurs. Ses scénarios sont les scénarios gagnants. Un scénario qui n'est dans aucune stratégie gagnante est un scénario perdant. Un scénario est en conflit avec le CSP sous-jacent si $(\bigwedge_{1 \leq i \leq n} v_i = val_i) \wedge (\bigwedge_{c_j \in \mathbf{C}} c_j)$ est faux (au moins une contrainte est violée). Un scénario peut-être à la fois prometteur et perdant.

Nous pouvons donner une sémantique plus intuitive, récursive, pour la définition d'une stratégie gagnante d'un QCSP. Prenons un QCSP QC, deux cas sont possibles : soit la première variable est quantifiée universellement et nous avons le QCSP $\forall x QC$, soit elle est quantifiée existentiellement et nous avons le QCSP $\exists x QC$. Le QCSP $\forall x QC$ avec $x \in D_x$ admet une stratégie gagnante si et seulement si, pour tout $val \in D_x$, $Q(C \wedge (x = val))$ admet une stratégie gagnante. Le QCSP $\exists x QC$ avec $x \in D_x$ admet une stratégie gagnante si et seulement si, pour au moins un $val \in D_x$, $Q(C \wedge (x = val))$ admet une stratégie gagnante.

Par exemple, la stratégie représentée à la figure 1 est une stratégie gagnante pour le QCSP

$$\exists x \exists y \forall z \exists t((x = (y * z) + t) \wedge (t \leq x)), x, y, z, t \in \{0, 1, 2\}$$

puisque $(0 = (0 * 0) + 0)$, $(0 = (0 * 1) + 0)$ et $(0 = (0 * 2) + 0)$. Le scénario $0, 0, 2, 0$, qui correspond à l'instanciation complète $(x = 0)$, $(y = 0)$, $(z = 2)$ et $(t = 0)$, est un scénario prometteur puisque $0 = (0 * 2) + 0$ et est aussi un scénario gagnant puisque faisant partie de la stratégie gagnante ci-dessous. Le scénario $2, 2, 0, 2$, qui correspond à l'instanciation complète $(x = 2)$, $(y = 2)$, $(z = 0)$ et $(t = 2)$, est aussi un scénario prometteur puisque $2 = (2 * 0) + 2$ mais est un fait un scénario perdant

puisqu'il ne fait partie d'aucune stratégie gagnante ($\exists t((2 = (2 * 1) + t) \wedge (t \leq x)), t \in \{0, 1, 2\}$ ne peut avoir de solution). Le scénario $2, 2, 2, 2$ est en conflit avec le CSP $((x = (y * z) + t) \wedge (t \leq x))$.

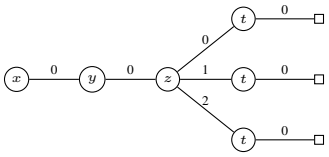


Figure 1. Exemple de stratégie gagnante

Nous utilisons aussi la représentation sous forme d'un QCSP+. En effet, un QCSP+ recouvre syntaxiquement les QCSP de la forme

$$
\begin{array}{l}
Q\phi = \exists x_1 \forall y_1 \ldots \exists x_{n-1} \forall y_{n-1} \exists x_n \\
R_\exists(x_1) \wedge (R_\forall(x_1, y_1) \rightarrow \\
\quad (\ldots \quad (R_\exists(x_1, y_1, \ldots, x_{n-1}) \wedge \\
\qquad\quad (R_\forall(x_1, y_1, \ldots, x_{n-1}, y_{n-1}) \rightarrow \\
\qquad\quad (R_\exists(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \wedge \\
\qquad\quad F(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n)))) \ldots))
\end{array}
$$

avec $x_1 \in D_{x_1}, y_1 \in D_{y_1}, \ldots, x_{n-1} \in D_{x_{n-1}}, y_{n-1} \in D_{y_{n-1}}, x_n \in D_{x_n}$.

Cela peut apparaître comme une restriction, mais il n'en n'est rien puisqu'il suffit de mettre $R_\exists(x_1, y_1, \ldots, x_i) \equiv \top$ pour tout i, $i \leq n$, et $R_\forall(x_1, y_1, \ldots, x_i, y_i) \equiv \top$ pour tout i, $i < n$, pour obtenir à nouveau un QCSP quelconque $QF(x_1, \ldots, x_n)$.

3. Exemples de jeux à deux joueurs à horizon fini modélisés en QCSP

Parmi les jeux à horizon fini, certains sont bien adaptés pour être modélisés par un QCSP. En effet, le large choix de contraintes disponibles dans une bibliothèque CSP permet de modéliser les règles et les conditions de victoires de jeux complexes. Le lieur lui permet de représenter chaque joueur ainsi que l'alternance des tours de jeux. Nous présentons ici trois jeux : le *Jeu de Nim*, le *MatrixGame* et le *Puissance Quatre*. Ces trois jeux sont sélectionnées car ils permettent d'apprécier différentes caractéristiques des QCSP. Le *Jeu de Nim* est un jeu dans lequel des règles doivent être satisfaites entre chaque tour de jeu. Son modèle en QCSP fait donc apparaître une hiérarchie de règles qui est prise en compte lors de la résolution. Le *MatrixGame* s'exprime très simplement grâce à l'utilisation d'une contrainte globale, il démontre l'intérêt d'un formalisme puissant de haut niveau tel que les CSP. Enfin, le *Puissance Quatre* est un jeu populaire qui nécessite un grand nombre de variables et de contraintes pour être modélisé. Bien que difficile à résoudre pour la taille de grille usuelle, c'est un excellent challenge pour solveur QCSP.

3.1. Le Jeu de Nim

Le *Jeu de Nim* est un jeu à deux joueurs qui se joue avec un tas de pièces ou d'allumettes. Le but du jeu est d'être celui qui se saisira de la dernière allumette. Chaque joueur peut prendre de une à trois allumettes. Dans la variante de Fibonacci, le minimum est d'une allumette et le maximum pour le premier tour est du nombre initial d'allumettes moins une. Puis chaque joueur peut prendre d'une jusqu'au double d'allumettes que son adversaire à pris au tour précédent. Le joueur existentiel débute. Avec un nombre pair n d'allumettes, le QCSP est le suivant (x_i est le nombre d'allumettes choisies au tour i): $R_\exists(1) = \top$ (Le joueur existentiel choisit entre 1 et $n-1$ allumettes, toutes les valeurs du domaine de x_1 sont possibles) et pour tout i, $1 < i \leq \frac{n}{2}$, $R_\exists(i) = (x_i \leq 2 * y_{i-1}) \wedge (x_i + \Sigma_{1 \leq j < i}(x_j + y_j) \leq n)$ et pour tout i, $1 \leq i \leq \frac{n}{2}$, $R_\forall(i) = (y_i \leq 2 * x_i) \wedge (\Sigma_{1 \leq j \leq i}(x_j + y_j) \leq n)$ (chaque joueur prend d'une jusqu'au double d'allumettes que son adversaire à pris au tour précédent et ne peut pas prendre plus d'allumettes que ce qu'il reste dans le tas) :

$$\exists x_1 \forall y_1 \ldots \exists x_{n-1} \forall y_{n-1} \\ R_\exists(1) \wedge (R_\forall(1) \rightarrow (R_\exists(2) \wedge (R_\forall(2) \rightarrow (\ldots (R_\forall(n-1) \rightarrow \bot)))))$$

avec $x_1, y_1, \ldots, x_{n-1}, y_{n-1} \in [1..n-1]$.

Pour $n = 4$, nous obtenons le QCSP:

$$\begin{array}{l} \exists x_1 \forall y_1 \exists x_2 \forall y_2 \\ \quad (((y_1 \leq 2 * x_1) \wedge ((x_1 + y_1) \leq 4)) \rightarrow \\ \quad (((x_2 \leq 2 * y_1) \wedge ((x_1 + y_1 + x_2) \leq 4)) \wedge \\ \quad (((y_2 \leq 2 * x_2) \wedge ((x_1 + y_1 + x_2 + y_2) \leq 4)) \rightarrow \bot))) \\ \quad \text{avec } x_1, y_1, x_2, y_2 \in \{1, 2, 3\} \end{array} \quad (1)$$

La figure 2 montre la première stratégie gagnante (sous forme d'arbre) obtenue à partir de l'arbre de recherche de la figure 3 par une simple application de la sémantique des quantificateurs.

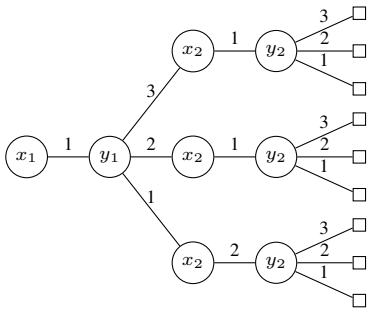


Figure 2. Première stratégie gagnante pour le problème Nim avec la variante de Fibonacci ($n = 4$)

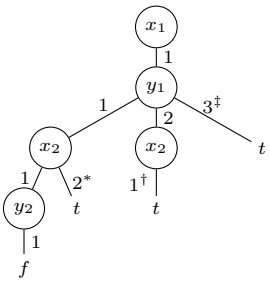


Figure 3. Arbre de recherche pour le problème Nim avec la variante de Fibonacci $(n = 4)$

Pour l'instanciation $(x_1 = 1), (y_1 = 1), (x_2 = 1)$, le QCSP est réduit à $\forall y_2(((y_2 \leq 2) \wedge (y_2 \leq 1)) \rightarrow \bot) \equiv \bot$ avec $y_2 \in \{1, 2, 3\}$ et pour l'instanciation $(x_1 = 1)$, $(y_1 = 1)$, $(x_2 = 2)$ ($*$ de la figure 3), le QCSP est réduit à $\forall y_2(((y_2 \leq 4) \wedge (y_2 \leq 0)) \rightarrow \bot) \equiv \top$ avec $y_2 \in \{1, 2, 3\}$.

Pour l'instanciation $(x_1 = 1), (y_1 = 2), (x_2 = 1)$ († de la figure 3), le QCSP est réduit à $\forall y_2(((y_2 \leq 2) \wedge (y_2 \leq 0)) \rightarrow \bot) \equiv \top$ avec $y_2 \in \{1, 2, 3\}$.

Pour l'instanciation $(x_1 = 1), (y_1 = 3)$ ($\ddagger$ de la figure 3), le QCSP est réduit à $\exists x_2 \forall y_2 \lambda \equiv \top$ avec $x_2, y_2 \in \{1, 2, 3\}$ et $\lambda = ((3 \leq 2) \rightarrow (((x_2 \leq 6) \wedge (x_2 \leq 0)) \wedge (((y_2 \leq 2 * x_2) \wedge ((x_2 + y_2) \leq 0)) \rightarrow \bot)))$. Pour chacune des instanciations partielles $*$, † et $\ddagger$, toutes les branches sont vraies.

3.2. Le MatrixGame

Le *MatrixGame* est un jeu à deux joueurs de d tours. Il est joué sur une matrice de $0/1$ de taille 2^d. À chaque tour, le joueur existentiel coupe horizontalement la matrice en deux et décide s'il faut conserver la partie haute ou basse. Puis, le joueur universel coupe la matrice verticalement et décide s'il faut conserver la partie gauche ou droite. Le joueur existentiel gagne si la dernière case contient 1.

Le *MatrixGame* est modélisé de la manière suivante :

$$\text{Soit le lieur} \quad \exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n \quad x_i \in \{0, 1\} \tag{1}$$

$$\text{alors} \quad Board[Idx] = 1 \tag{2}$$

$$\text{avec} \quad Idx = \sum_{i=1}^{n} access_i x_i, \quad Idx \in [0; l] \tag{3}$$

$$l = 2^d * 2^d$$

$$n = 2 * d$$

Le lieur (1) définit les variables de décision du problème, chacune prend sa valeur dans $\{0, 1\}$. Une alternance de quantificateurs correspond à un tour de jeu. Ainsi, pour un problème de d-tours, il y a $n = 2 * d$ variables de décisions et la matrice de jeu comporte $l = 2^{d^2}$ cases. La matrice de jeu est l'unique entrée du problème. Chacune de ses cases étant égale soit à 0 soit à 1. C'est le vecteur $Board$ utilisé dans la contrainte (2) qui représente la matrice de jeu dans notre modèle. La variable Idx représente l'ensemble des indices des cases encore accessibles d'ici la fin du jeu. Pour gagner, le joueur existentiel doit donc vérifier la contrainte 2 qui assure qu'il reste au moins une case égale à 1 dans le reste des cases atteignables, et ce jusqu'au dernier tour. Dans un CSP, cette relation est modélisée grâce à la contrainte `element`.

Le vecteur $access$ utilisé dans la contrainte (3) permet de connecter les variables de décision aux valeurs de la matrice de jeu. Il permet en effet de retrouver la case finale (la dernière restante) de la matrice de jeu en fonction des variables de décision x_i. Il est calculé avant la résolution du problème, il est donc constant pour celui-ci. Sa taille correspond au nombre de variables de décision du problème (2 au minimum). Il est calculé de la manière suivante :

$$access_n = 1$$
$$access_{n-1} = l$$
$$access_i = access_{i+2} * 2$$

Pour un jeu à 3 tours, nous obtenons le vecteur suivant :

32	4	16	2	8	1

Ainsi, il est possible de calculer l'indice de la case finale. Soit la partie réalisée par la séquence de coupes (0,1,1,0,1,1), l'indice (modélisée par la variable Idx) de la case finale est 29, ce qui correspond à la case de coordonnées ligne 4 et colonne 5. Si la valeur de cette case est égale à 1, le joueur existentiel gagne. Cette partie correspond à un scénario prometteur.

Résoudre le QCSP du *MatrixGame* signifie découvrir une stratégie gagnante pour ce jeu. Cette stratégie permet au joueur existentiel de gagner quelques soient les coups de son adversaire. Dans ce jeu, il n'y a pas de règle à vérifier au cours de la partie comme pour le *Jeu de Nim*. Seule la valeur de la dernière case est pertinente. Ainsi, l'arbre de recherche est complet et nécessite d'explorer complètement tous les scénarios.

3.3. Le Puissance Quatre

Le jeu du *Puissance Quatre* est un jeu bien connu qui se joue à deux joueurs sur une grille comptant le plus souvent 6 rangées et 7 colonnes. Le joueur qui gagne la

partie est le premier à aligner 4 de ses pions (horizontalement, verticalement ou en diagonal). Tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, le pion descend alors jusqu'à la position la plus basse disponible dans la colonne. Si toute la grille est remplie et qu'aucun joueur n'a réalisé d'alignement, la partie est déclarée nulle.

La modélisation donnée ici peut être utilisée pour un nombre variables de lignes et de colonnes variable mais supérieur à quatre (le nombre minimum de pions à aligner). Le nombre de colonnes sera appelé nbC et le nombre de lignes nbL. Elle est traduite et adaptée de celle de P. Nightingale dans (Nightingale, 2009). L'approche est la suivante : à chaque tour de jeu est associée une grille représentant l'état du jeu à ce tour, celle-ci sera liée aux grilles des tours précédents et suivants. Chaque case d'une grille peut prendre trois états : libre, occupée par le joueur 1 ou occupée par le joueur 2. Pour assurer la cohérence des grilles au fur et à mesure des tours de jeux, des contraintes de liaisons seront ajoutées au modèle. Ainsi, si au tour de jeu k, la case de coordonnées $(2, 4)$ est occupée par le joueur 1, dans toutes les grilles des tours $\geq k$, la case de coordonnées $(2, 4)$ sera également occupée par le joueur 1.

Le nombre de variables de décision correspond au nombre de cases dans la grille de départ, soit $nbL \times nbC$ variables chacune à valeur dans l'intervalle $[1..nbC]$. Le lieur est formé d'une alternance de quantificateurs $\exists\forall$ fonction du nombre de variables de décisions. Nous obtenons le problème suivant :

$$\text{Soit le lieur} \quad \exists m^1 \forall u^2 \dots \exists m^{n-1} \forall u^n \tag{4}$$

$$\text{avec } \{m^i, u^i\} \in [1 \dots nbC] \tag{5}$$

$$i \in [1 \dots nbL \times nbC] \tag{6}$$

$$\text{tel que} \quad \mathcal{C} \tag{7}$$

Ici, $\mathcal{C}$ représente l'ensemble des contraintes du problème à satisfaire.

Pour chaque tour de jeu, donc pour chaque i, il est nécessaire d'introduire un ensemble de variables intermédiaires comme suit :

– Si i est pair (tour du joueur $\forall$), alors ajout d'une nouvelle variable $\exists m^i$ qui sera liée à la variable $\forall u^i$ correspondante.

– $\exists b^i_{r,c} \in \{rond, croix, vide\}$ avec $r \in [1 \dots nbL]$ et $c \in [1 \dots nbC]$. Chaque $b^i_{r,c}$ représente l'état du tableau après le mouvement i.

– $\exists h^i_c \in [0 \dots nbL], c \in [1 \dots nbC]$ représentant la hauteur de la colonne c après le tour i.

– $\exists vainqueur^i \in \{rond, croix, aucun\}$ représentant le gagnant après le tour i.

– $\exists aligne^i \in \{0, 1\}$ représentant la présence d'un alignement après le tour i.

– $\exists al^i_z \in \{0, 1\}$ représentant la présence d'un alignement dans chaque, ligne, colonne ou diagonale, numérotée z après le tour i (un alignement ne concerne que les

pions du joueur qui joue le tour i).

– $\exists mh_c^i \in \{0,1\}, c \in [1 \dots nbC]$, qui est égale à 1 quand le pion au tour i a été placé dans la colonne c.

– $\exists pos_{r,c}^i \in \{0,1\}, r \in [1 \dots nbL], c \in [1 \dots nbC]$, qui est égale à 1 quand la case de coordonnées (r,c) est libre pour le tour i.

– $\exists tour\_fini^i \in \{0,1\}$ qui est égal à 1 quand le tour est terminé et que toutes les contraintes du tour sont satisfaites.

Ces variables sont liées entre elles ainsi qu'aux variables de décisions grâce aux 7 groupes de contraintes suivants :

1. Si i est pair (tour du joueur $\forall$), la contrainte suivante est ajoutée pour chaque colonne c : $(vainqueur^{i-1} \neq aucun) \vee (h_c^{i-1} = nbL) \vee ((u^i = c) \rightarrow (m^i = c))$

2. Si i est pair, un pion $croix$ est placé dans la grille. Pour chaque colonne c, la valeur de la variable mh_c^i est choisie en fonction du mouvement effectué par le joueur (a-t-il été effectué dans cette colonne ou non). Pour cela, l'ensemble des contraintes suivantes est ajouté :

- $((m^i = c) \wedge (h_c^{i-1} \neq nbL)) \leftrightarrow ((mh_c^i = 1) \vee (tour\_fini^i = 1))$
- Pour chaque $r \in [1 \dots nbL], c \in [1 \dots nbC]$:

remplir la colonne complètement

$(h_c^{i-1} = r-1) \rightarrow (pos_{r,c}^i = 1)$

$(b_{r,c}^i \neq rond \wedge b_{r+1,c}^i = aucun \wedge b_{r+2,c}^i = aucun \wedge \dots \wedge b_{nbL,c}^i = aucun) \Leftrightarrow (pos_{r,c}^i = 1)$

connecter mh_c^i *à* $b_{r,c}^i$

$(mh_c^i = 1 \wedge h_c^{i-1} = r-1) \rightarrow (b_{r,c}^i = croix)$

$(mh_c^i \neq 1 \wedge h_c^{i-1} = r-1) \rightarrow (b_{r,c}^i = aucun)$

Si i est impair, des contraintes analogues sont ajoutées en inversant $rond$ et $croix$.

3. Les contraintes suivantes assurent la cohérence entre la grille du tour $i-1$ et la grille du tour i : pour chaque $r \in [1 \dots nbL], c \in [1 \dots nbC]$, $((b_{r,c}^{i-1} = rond) \rightarrow (b_{r,c}^i = rond)) \wedge ((b_{r,c}^{i-1} = croix) \rightarrow (b_{r,c}^i = croix))$

4. Les contraintes suivantes assurent la cohérence entre la hauteur d'une colonne et la grille de jeu : pour chaque $r \in [1 \dots (nbL+1)], c \in [1 \dots nbC]$: $(b_{r-1,c}^i \neq aucun \wedge b_{r,c}^i = aucun) \rightarrow (h_c^i = r-1)$.

5. Détection d'un alignement : soit bg_z un groupe numéroté z de quatre variables formant un alignement de 4 pions (lignes, colonnes ou diagonales). Par exemple, le groupe $bg_1 = (b_{1,1}^i, b_{2,1}^i, b_{3,1}^i, b_{4,1}^i)$). Alors $\max(z) = nbL + nbC + 2*(nbL-3) + 2*(nbC-3) - 2$. Si i est impair, pour tout z, $(aligne^{i-1} = 1 \vee bg_z[1] \neq rond \vee bg_z[2] \neq rond \vee bg_z[3] \neq rond \vee bg_z[4] \neq rond) \Leftrightarrow al_z^i \neq 1$. Si i est pair, des contraintes analogues sont ajoutées en inversant $rond$ et $croix$.

6. Détection globale d'un alignement au tour i : $l_1^i \vee l_2^i \vee \dots \vee l_{\max(z)}^i \Leftrightarrow aligne^i$.

7. Si i est impair, les contraintes suivantes assurent la détermination du vainqueur :

- $(vainqueur^{i-1} = rond) \rightarrow (vainqueur^i = rond)$
- $(vainqueur^{i-1} = croix) \rightarrow (vainqueur^i = croix)$
- $(vainqueur^{i-1} = aucun \wedge aligne^i = 1) \rightarrow (vainqueur^i = rond)$
- $(vainqueur^{i-1} = aucun \wedge aligne^i = 0) \rightarrow (vainqueur^i = aucun)$.

Si i est pair, des contraintes analogues sont ajoutées en inversant $rond$ et $croix$.

Certaines contraintes font référence à celles du tour précédent, pour les contraintes du premier tour (pour $i = 1$), les valeurs suivantes sont utilisés : $vainqueur^0 = aucun, al^0 = 0, b^0_{r,c} = aucun, b^i_{0,c} = rond, h^0_c = 0, tour\_fini_0 = 1$. Il faut de plus assurer que le joueur $\exists$ remporte le jeu, donc $vainqueur^{nbL \times nbC} = rond$. Pour le premier tour, la symétrie est cassée en retirant la partie gauche du domaine de départ. Ainsi, le domaine de la variable $\exists m_1$ devient $\{\lfloor \frac{nbC}{2} \rfloor \dots nbC\}$.

4. Un solveur QCSP comme extension d'un solveur CSP

Pour un solveur QCSP conçu comme une extension d'un solveur CSP, l'ensemble des contraintes est considéré comme une conjonction. Quand un problème est représenté pour être une entrée d'un tel solveur QCSP, les contraintes sont issues d'un ensemble de contraintes primitives du langage. Cependant un problème représentant un jeu à deux joueurs est rarement exprimé comme tel mais plutôt par une grande formule logique, que nous appellerons QCSP *complexe* (ie. contenant des contraintes booléennes qui ne sont pas des conjonctions). Par exemple, dans un jeu fini à deux joueur sur n tours, le problème est exprimé en deux parties. Il commence par une alternance de vérifications des règles pour le joueur existentiel ($R_\exists$) d'une part et pour son adversaire, le joueur universel, ($R_\forall$) d'autre part. Il termine par une vérification des conditions de victoire pour le joueur existentiel (VC):

$$
\begin{array}{l}
\exists x_1 \forall y_1 \dots \exists x_{n-1} \forall y_{n-1} \exists x_n \\
R_\exists(x_1) \wedge (R_\forall(x_1, y_1) \rightarrow \\
\quad (\dots \quad (R_\exists(x_1, y_1, \dots, x_{n-1}) \wedge \\
\qquad (R_\forall(x_1, y_2, \dots, x_{n-1}, y_{n-1}) \rightarrow \\
\qquad (R_\exists(x_1, y_2, \dots, x_{n-1}, y_{n-1}, x_n) \wedge \\
\qquad VC(x_1, y_2, \dots, x_{n-1}, y_{n-1}, x_n)))) \dots))
\end{array} \tag{8}
$$

Pour transformer ce QCSP complexe en une entrée pour un solveur QCSP, des symboles propositionnels existentiellement quantifiés (dits de Tseitin) sont introduits. Ainsi le QCSP du *Jeu de Nim* (1) est réécrit en :

$$\begin{array}{l}\exists x_1 \forall y_1 \exists x_2 \forall y_2 \exists o_1 \exists o_2 \\ \quad (((y_1 \leq 2 * x_1) \wedge ((x_1 + y_1) \leq 4)) \rightarrow o_1) \wedge \\ \quad (o_1 \leftrightarrow (((x_2 \leq 2 * y_1) \wedge ((x_1 + y_1 + x_2) \leq 4)) \wedge o_2)) \wedge \\ \quad (o_2 \leftrightarrow (((y_2 \leq 2 * x_2) \wedge ((x_1 + y_1 + x_2 + y_2) \leq 4)) \rightarrow \bot)) \\ \quad \text{avec } x_1, y_1, x_2, y_2 \in \{1, 2, 3\}, o_1, o_2 \in \textbf{BOOL}\end{array} \quad (9)$$

Algorithme 1 Un algorithme de recherche quantifié pour solveur de QCSP

Entrée: Un QCSP QC
Sortie: **vrai** si le QCSP QC admet au moins une stratégie gagnante et **faux** sinon

```
pileRetourArrière := ∅
tant que true faire
 selon atteintPointFixe(C) faire
  cas échec
   retour sur le dernier choix existentiel (x, k) de pileRetourArrière
   si aucune alors retourner faux
   sinon ajouter à C la contrainte (x = k)
  cas succès
   retour sur le dernier choix universel (x, k) de pileRetourArrière
   si aucune alors retourner vrai
   sinon ajouter à C la contrainte (x = k)
  cas branche
   sélectionner la variable non instanciée suivante x
   pour tout k ∈ D(x) empiler (x, k) dans pileRetourArrière
   sélectionner le premier choix (x, k) de pileRetourArrière
   ajouter à C la contrainte (x = k)
 fin selon
fin tant que
```

Nous verrons dans la section suivante que cette transformation n'affecte pas uniquement la modélisation du problème mais aussi le parcours de l'espace de recherche associé. Pour l'instant nous décrivons l'algorithme 1 qui présente la structure d'un algorithme de recherche quantifié pour un solveur QCSP conçu comme une extension d'un solveur CSP. Cet algorithme est basé sur une boucle perpétuelle. Au départ le pile de retour arrière portant sur le choix des variables est vide. La fonction $reachfixpoint$ calcule le point fixe de la propagation et retourne trois différents codes possibles : *échec* s'il y a une contrainte violée, $succès$ si l'ensemble des contraintes est vrai et $branche$ sinon. Dans le cas de *échec*, les derniers choix sur les variables universellement quantifiées sont sautés car ils ne peuvent pas mener à un succès, le choix sur la dernière variable existentiellement quantifiée (x, k) est dépilé de la pile de retour arrière et la nouvelle contrainte $(x = k)$ est insérée dans l'ensemble de contraintes courant pour explorer cette nouvelle possibilité pour la variable x existentiellement quantifiée. S'il n'y a plus de choix sur une quelconque variable existentiellement quantifiée (il ne reste plus que des choix sur des variables universellement

quantifiées ou plus du tout) alors le QCSP n'a pas de stratégie gagnante et l'algorithme retourne faux. Dans le cas de $succès$, les derniers choix sur les variables existentiellement quantifiées sont sautés car un seul succès est suffisant de par la sémantique du quantificateur existentiel, le choix sur la dernière variable universellement quantifiée (x, k) est dépilé de la pile de retour arrière et la nouvelle contrainte $(x = k)$ est insérée dans l'ensemble de contraintes courant pour continuer à construire une stratégie gagnante pour toutes les valeurs possibles de x. S'il n'y a plus de choix sur une quelconque variable universellement quantifiée (il ne reste plus que des choix sur des variables existentiellement quantifiées ou plus du tout) alors le QCSP a une stratégie gagnante et l'algorithme retourne vrai. Sinon, l'appel à $reachfixpoint(C)$ a retourné $branche$: l'algorithme continue d'avancer à travers le lieur Q et sélectionne la prochaine variable x non instanciée et empile sur la pile de retour arrière tous les choix de la variable. Le premier choix de la variable (x, k) est dépilé de la pile de retour arrière et la nouvelle contrainte $(x = k)$ est insérée dans l'ensemble de contraintes pour explorer la première possibilité de la variable x.

5. Solveur QCSP et jeux à deux joueurs à horizon fini

La plupart des solveurs de l'état de l'art, quasiment tous basés sur l'introduction des symboles propositionnels de Tseitin, ont un inconvénient majeur, le « talon d'Achille » : ils parcourent un espace combinatoire bien plus grand que l'espace de recherche du problème original [1] qui est su comme étant inutile par construction (ce qui est le cas par exemple pour les sous-arbres $*$, † et ‡ de l'arbre de recherche de la figure 4 pour le solveur QCSP sur le *Jeu de Nim*). Cette notion inclut la capture des actions illégales du joueur universel mais aussi, par exemple, la détection de la fin du jeu avant le dernier tour qui est aussi une source d'accroissement de l'espace de recherche exploré [2].

Même si le QCSP complexe et le QCSP issu de l'introduction de symboles propositionnels de Tseitin sont équivalents, le second est bien plus complexe à résoudre car de nombreuses instances satisfiables sont explorées durant la recherche. Ces branches sont alors combinées (au niveau des variables universellement quantifiées) pour cons-

1. Le problème de l'exploration d'un espace de recherche trop grand a été déjà pointé du doigt dans la communauté QBF (Ansotegui *et al*., 2005) : Dans le cas d'un jeu à deux joueurs, les contraintes booléennes deviennent nécessairement vraies grâce à une action illégale du joueur universel qui conduit immédiatement à la victoire du joueur existentiel. Dans leur enfance, les solveurs QBF avaient du mal à détecter que les contraintes booléennes étaient nécessairement vraies sous une assignation partielle. Comme les solveurs QBF sont basés pour la plupart sur un algorithme de recherche quantifié pouvant être vu comme une extension de l'algorithme de recherche des solveurs SAT, les premiers solveurs QBF étaient quasiment tous basés sur des solveurs SAT existants. Ces solveurs QBF ont traités principalement le problème du « talon d'Achille » en changeant le cœur du solveur (introduisant par exemple des variables indicatrices qui capturent les actions illégales du joueur universel (Ansotegui *et al*., 2005), exploitant la représentation sous forme de circuit (Goultiaeva, Bacchus, 2010) ou plus généralement tenant compte de la dualité des problèmes QBF (Sabharwal *et al*., 2006)).

2. Dans (Ansotegui *et al*., 2005), si le joueur existentiel a déjà gagné alors toute nouvelle action du joueur universel est considérée comme illégale.

truire des stratégies gagnantes. Une propriété de la logique sous-jacente très importante pour l'efficacité est perdue dans le solveur : la propriété d'absorption du $\top$ par rapport à la disjonction i.e. si un argument d'une disjonction n-aire est vrai alors il est inutile de calculer les autres arguments, la disjonction est vraie. Si $R(x_1, \ldots, x_i)$ est vrai pour une instanciation quelconque alors il n'est pas nécessaire de résoudre $q_{x_{i+1}} x_{i+1} \ldots q_{x_n} x_n F(x_1, \ldots, x_n)$ pour cette instanciation mais le solveur le fait. La figure 4 l'illustre en explicitant l'arbre de recherche pour le *Jeu de Nim* selon le QCSP (9).

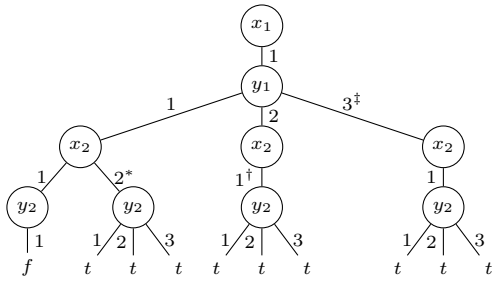


Figure 4. Arbre de recherche pour le problème Nim avec la variante de Fibonacci $(n = 4)$ *pour un solveur QCSP*

Par construction, la propriété d'absorption de $\bot$ par rapport à la conjonction est assurée par un solveur (Q)CSP : si une contrainte est fausse, puisque la file des contraintes est une conjonction n-aire, il n'est pas nécessaire de calculer les autres contraintes, le solveur échoue puis revient en arrière.

Dans un solveur pour QCSP, la propriété d'absorption de $\top$ par rapport à la disjonction n'est plus assurée. Même si pour une instanciation quelconque la conjonction de contraintes est vraie (et donc $((o_1 \vee o'_1) \leftrightarrow \top)$ est aussi vrai), le solveur QCSP tentera de résoudre le QCSP $q_{x_{i+1}} x_{i+1} \ldots q_{x_n} x_n \exists o_1 \ldots \exists o_m C(x_1, \ldots, x_n)$ pour cette instanciation (où C est l'ensemble de contraintes issue de F par introduction des symboles propositionnels o_1, …, o_m) . En d'autres termes, pour détecter un succès, dans un solveur QCSP, toutes les contraintes doivent être satisfaites et tant qu'il demeure des variables non instanciées ou des contraintes non résolues, l'algorithme 1 de recherche quantifié ne peut pas décider.

Si la puissance d'un solveur QBF relève principalement de l'algorithme de recherche quantifié et de sa représentation interne (incluant des techniques d'apprentissage de clauses et de cubes), la puissance d'un solveur (Q)CSP relève plus de son catalogue de contraintes. Mais il apparaît bien plus difficile d'implanter les techniques liés à la dualité dans les solveurs QCSP que dans les solveurs QBF puisqu'il faut alors disposer des contraintes duales. Ainsi, il nous est apparu utile de construire un solveur QCSP au-dessus des technologies CSP *sans les toucher*. Pour traiter le problème du « talon d'Achille » sans toucher à la librairie CSP, nous proposons un outil très simple pour les solveurs QCSP, l'*outil coupure*, inspiré de la coupure de Prolog, comme étant un outil sous la responsabilité de celui qui spécifie le QCSP, pour élaguer l'espace de

recherche des parties qui sont par construction inutiles. Cet outil ne demande qu'une seule chose : que nous soyons capable de dire au solveur QCSP si le QCSP courant est résolu.

L'outil *coupure* est défini, pour un ensemble de contraintes S et une contrainte e comme $coupure(S, e)$: si $\neg(\bigwedge_{c \in S} c \rightarrow e)$ est vrai alors le QCSP courant est résolu.

Le théorème suivant met en exergue que cet outil est particulièrement puissant dans les jeux à deux joueurs à horizon fini puisqu'il permet d'éliminer le « talon d'Achille » : les stratégies gagnantes sont conservées (premier point du théorème) alors que l'espace de recherche du QCSP initial est recouvré (second point du théorème).

THÉORÈME 1. — *Soit $Q\phi$ le QCSP complexe suivant spécifiant un jeu à deux joueurs :*

$$\begin{array}{l} Q\phi = \exists x_1 \forall y_1 \ldots \exists x_{n-1} \forall y_{n-1} \exists x_n \\ R_\exists(x_1) \wedge (R_\forall(x_1, y_1) \rightarrow \\ \quad (\ldots \quad (R_\exists(x_1, y_1, \ldots, x_{n-1}) \wedge \\ \qquad\quad (R_\forall(x_1, y_1, \ldots, x_{n-1}, y_{n-1}) \rightarrow \\ \qquad\quad (R_\exists(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \wedge \\ \qquad\quad VC(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n)))) \ldots)) \end{array}$$

avec $x_1 \in D_{x_1}, y_1 \in D_{y_1}, \ldots, x_{n-1} \in D_{x_{n-1}}, y_{n-1} \in D_{y_{n-1}}, x_n \in D_{x_n}$.

Soient C l'ensemble de contraintes obtenu de la réécriture de la formule ϕ par l'introduction des symboles propositionnels de Tseitin o_1, …, o_m et l'ensemble

$$\begin{array}{l} \Sigma = \{coupure(\{R_\forall(x_1, y_1), \ldots, R_\forall(x_1, y_1, \ldots, x_{i-1}, y_{i-1})\}, \\ \quad (R_\exists(x_1, y_1, \ldots, x_i) \rightarrow R_\forall(x_1, y_1, \ldots, x_i, y_i)) \ ) \mid 1 \leq i < n \ \} \end{array}$$

de coupures. Les propositions suivantes sont vérifiées ($Q_\exists = \exists o_1 \ldots \exists o_m$) :

– Soit s une stratégie pour $QQ_\exists$. s est une stratégie gagnante pour $QQ_\exists C$ si et seulement si s est une stratégie gagnante pour $QQ_\exists(C \wedge \Sigma)$.

– Soit T un arbre de recherche pour Q. T est un arbre de recherche de $Q\phi$ si et seulement si T est un arbre de recherche pour $QQ_\exists(C \wedge \Sigma)$.

PREUVE. — Le premier point du théorème est une conséquence immédiate du second puisque si les arbres de recherche sont identiques, les stratégies gagnantes aussi. Prouvons le second point. Soit la propriété P suivante : « T' est un sous arbre de hauteur $2i$ d'un arbre de recherche pour $Q\phi$ si et seulement si T' est aussi un sous arbre d'un arbre de recherche pour $QQ_\exists(C \wedge \Sigma)$ ». La preuve se fait sur les alternances de quantificateurs $\exists\forall$. Nous notons $R_\exists(x_1, y_1, \ldots, x_k)$ et $R_\forall(x_1, y_1, \ldots, x_k, y_k)$ respectivement $R^k_\exists$ et $R^k_\forall$.

Supposons $i = 1$. Si $R^1_\exists$ est vrai, soit $R^1_\forall$ est vrai alors le nœud est ouvert (la branche n'est pas décidée) aussi bien dans T_ϕ (l'arbre de recherche pour $Q\phi$) que $T_{(C \wedge \Sigma)}$ (l'arbre de recherche pour $QQ_\exists(C \wedge \Sigma)$) car dans ce dernier cas la coupure $coupure(\emptyset, (R^1_\exists \rightarrow R^1_\forall))$ n'est pas appliquée ($(R^1_\exists \rightarrow R^1_\forall) \equiv \top$), soit $R^1_\forall$ est faux et

alors c'est un nœud fermé et $succès$ aussi bien dans T_ϕ car $(R^1_\exists \wedge (R^1_\forall \rightarrow \psi)) \equiv \top$ que dans $T_{(C \wedge \Sigma)}$ car dans ce dernier cas la coupure $coupure(\emptyset, (R^1_\exists \rightarrow R^1_\forall))$ est appliquée ($(R^1_\exists \rightarrow R^1_\forall) \equiv \bot$). Si $R^1_\exists$ est faux alors le nœud est fermé et $échec$ aussi bien dans T_ϕ car $(R^1_\exists \wedge (R^1_\forall \rightarrow \psi)) \equiv \bot$ que dans $T_{(C \wedge \Sigma)}$ car dans ce dernier cas la coupure $coupure(\emptyset, (R^1_\exists \rightarrow R^1_\forall))$ n'est pas appliquée ($(R^1_\exists \rightarrow R^1_\forall) \equiv \top$).

Nous supposons que la propriété est vraie au rang $i-1$ et nous démontrons qu'elle reste vraie au rang i. Un nœud dans le sous arbre est soit ouvert soit fermé et dans ce dernier cas, le résultat est alors $succès$ ou $échec$. Par hypothèse de récurrence, T' est un sous arbre de hauteur $2(i-1)$ d'un arbre de recherche T_ϕ pour $Q\phi$ si et seulement si T' est aussi un sous arbre d'un arbre de recherche $T_{(C \wedge \Sigma)}$ pour $QQ_\exists(C \wedge \Sigma)$. Les lieurs sur Q étant identiques et l'algorithme de construction des arbres étant le même un nœud est ouvert dans l'un si et seulement s'il est ouvert dans l'autre. Ce qui signifie que pour tout k, $k < i$, les $R^k_\forall$ et les $R^k_\exists$ sont vrais. Au rang i, seule la coupure $coupure(\{R^1_\forall, \ldots, R^{i-1}_\forall\}, (R^i_\exists \rightarrow R^i_\forall))$ peut s'appliquer. Maintenant un nœud de profondeur $2i$ est un nœud soit :

(ouvert) : $\Rightarrow$ Si le nœud est un nœud ouvert de T_ϕ alors $R^i_\exists$ et $R^i_\forall$ sont vrais donc la coupure $coupure(\{R^1_\forall, \ldots, R^{i-1}_\forall\}, (R^i_\exists \rightarrow R^i_\forall))$ ne s'applique pas (car $(R^i_\exists \rightarrow R^i_\forall) \equiv \bot$) donc le nœud est aussi un nœud ouvert dans $T_{(C \wedge \Sigma)}$.

$\Leftarrow$ Si le nœud est un nœud ouvert dans $T_{(C \wedge \Sigma)}$ alors $R^i_\exists$ et $R^i_\forall$ sont vrais donc le nœud est aussi un nœud ouvert dans T_ϕ.

(fermé et $succès$) : $\Rightarrow$ Si le nœud est un nœud fermé et $succès$ dans T_ϕ alors $R^i_\exists$ est vrai et $R^i_\forall$ est faux donc la coupure $coupure(\{R^1_\forall, \ldots, R^{i-1}_\forall\}, (R^i_\exists \rightarrow R^i_\forall))$ s'applique puisque ($(R^i_\exists \rightarrow R^i_\forall) \equiv \bot$) donc le nœud est aussi fermé et $succès$ dans $T_{(C \wedge \Sigma)}$.

$\Leftarrow$ Si le nœud est un nœud fermé et $succès$ dans $T_{(C \wedge \Sigma)}$ alors c'est par application de la coupure $coupure(\{R^1_\forall, \ldots, R^{i-1}_\forall\}, (R^i_\exists \rightarrow R^i_\forall))$ donc $(R^i_\exists \rightarrow R^i_\forall) \equiv \bot$ donc $R^i_\forall$ est faux et $R^i_\exists$ est vrai donc le nœud est aussi un nœud fermé et $succès$ dans T_ϕ ($(R^i_\exists \wedge (R^i_\forall \rightarrow \psi)) \equiv \top$).

Enfin, un nœud de profondeur $2i-1$ est un nœud

(fermé et $échec$) $\Rightarrow$ Si le nœud est un nœud fermé et $échec$ dans T_ϕ alors une des contraintes de C est violée or la coupure $coupure(\{R^1_\forall, \ldots, R^{i-1}_\forall\}, (R^i_\exists \rightarrow R^i_\forall))$ est non applicable. En effet, tous les $\{R^1_\forall, \ldots, R^{i-1}_\forall\}$ sont vrais et $R^i_\exists$ est faux. C'est donc aussi un nœud fermé et $échec$ dans $T_{(C \wedge \Sigma)}$.

$\Leftarrow$ Si le nœud est un nœud fermé et $échec$ dans $T_{(C \wedge \Sigma)}$ alors une des contraintes de C est violée donc c'est aussi un nœud fermé et $échec$ dans T_ϕ.

■

Dans le cas d'un jeu à deux joueurs à horizon fini, chaque coupure

$$coupure(\quad \{R_\forall(x_1, y_1), \ldots, R_\forall(x_1, y_1, \ldots, x_{i-1}, y_{i-1})\}, \\ (R_\exists(x_1, y_1, \ldots, x_i) \rightarrow R_\forall(x_1, y_1, \ldots, x_i, y_i)))$$

s'exprime en logique ainsi :

$$\begin{array}{rlll} \neg(& (& \bigwedge_{1\leq k\leq i} R_\forall(x_1,y_1,\ldots,x_{k-1},y_{k-1})) \\ & \rightarrow & (R_\exists(x_1,y_1,\ldots,x_i) \rightarrow R_\forall(x_1,y_1,\ldots,y_{i-1},x_i,y_i)) \ ) \\ \equiv \quad (& (& \bigwedge_{1\leq k\leq i} R_\forall(x_1,y_1,\ldots,x_{k-1},y_{k-1})) \\ & \wedge & R_\exists(x_1,y_1,\ldots,x_i) \wedge \neg R_\forall(x_1,y_1,\ldots,y_{i-1},x_i,y_i) \ ) \end{array}$$

ce qui permet d'arrêter dès que le joueur universel triche (i.e. les joueurs ont respecté les règles jusqu'au coup i pour le joueur existentiel et $i-1$ pour le joueur universel puis le joueur universel a triché à son coup i).

Dans le cas de la modélisation du *Jeu de Nim*, le théorème demande d'ajouter deux nouvelles coupures :

$$\begin{array}{l} coupure(\emptyset, (\top \rightarrow ((y_1 \leq 2 * x_1) \wedge ((x_1 + y_1) \leq 4)))) \\ coupure(\quad \{(x_2 \leq 2 * y_1), ((x_1 + y_1 + x_2) \leq 4)\}, \\ \qquad\qquad (((x_2 \leq 2 * y_1) \wedge ((x_1 + y_1 + x_2) \leq 4)) \rightarrow \\ \qquad\qquad ((y_2 \leq 2 * x_2) \wedge ((x_1 + y_1 + x_2 + y_2) \leq 4)))) \end{array}$$

La première coupure élimine le sous arbre sous ‡ de la figure 4 :

$$[x_1 \leftarrow 1][y_1 \leftarrow 3](\neg((y_1 \leq 2 * x_1) \wedge ((x_1 + y_1) \leq 4)))$$

est vrai.

La seconde coupure élimine les deux sous arbres sous $*$ et † de la figure 4 : $C = (x_2 \leq 2 * y_1) \wedge ((x_1 + y_1 + x_2) \leq 4)$ et

$$\begin{array}{c} e = (((x_2 \leq 2 * y_1) \wedge ((x_1 + y_1 + x_2) \leq 4)) \rightarrow \\ ((y_2 \leq 2 * x_2) \wedge ((x_1 + y_1 + x_2 + y_2) \leq 4))) \end{array}$$

alors

$(*)$ $[x_1 \leftarrow 1][y_1 \leftarrow 1][x_2 \leftarrow 2](C)$ est vrai et $[x_1 \leftarrow 1][y_1 \leftarrow 1][x_2 \leftarrow 2](e)$ est faux puisque $y_2 \geq 0$ alors $[x_1 \leftarrow 1][y_1 \leftarrow 1][x_2 \leftarrow 2](\neg(C \rightarrow e))$ est vrai.

$(\dagger)$ $[x_1 \leftarrow 1][y_1 \leftarrow 2][x_2 \leftarrow 1](C)$ est vrai et $[x_1 \leftarrow 1][y_1 \leftarrow 2][x_2 \leftarrow 1](e)$ est faux puisque $y_2 \geq 0$ alors $[x_1 \leftarrow 1][y_1 \leftarrow 2][x_2 \leftarrow 1](\neg(C \rightarrow e))$ est vrai.

Cela permet de retrouver l'arbre de recherche de la figure 3.

Dans le cas du *Puissance Quatre*, le modèle présenté au paragraphe 3.3, bien que fonctionnel, a l'inconvénient de devoir compléter intégralement tous les tours de jeu pour déterminer le vainqueur même si celui-ci a été décidé plus tôt dans le jeu. Pour résoudre ce problème, il est possible d'utiliser la détection de valeurs pures (voir (Nightingale, 2009)). Ce problème peut aussi être résolu pour l'ajout de coupures. Ainsi, pour chaque tour de jeu i tel que i est pair, la coupure suivante est ajoutée : $coupure(\{vainqueur^i = rond, tour\_fini_i = 1\}, \bot)$. Cela garantit que la partie se termine dès que le vainqueur a été déterminé.

Le lien entre QCSP et jeux à deux joueurs a été présenté pour le cas où la stratégie calculée est une stratégie gagnante pour un jeu à horizon fini et à connaissance parfaite. Mais dans la pratique, une telle approche est intenable car soit il n'y a pas de stratégie gagnante initiale auquel cas le calcul réalisé n'est d'aucune aide, soit l'espace de recherche est tellement grand qu'il est illusoire de vouloir le parcourir, soit le jeu n'est pas à horizon fini ; voire les trois. Si nous disposions d'une fonction d'évaluation f pour une instanciation partielle, alors nous pourrions remplacer la condition de victoire $VC(x_1, y_1, \ldots, x_n)$ d'un QCSP par un test de franchissement de seuil s : $f(x_1, y_1, \ldots, x_n) \geq s$. La stratégie alors calculée n'est plus une stratégie gagnante (au sens du jeu) mais une stratégie qui garantit au joueur $\exists$ que quoique le joueur $\forall$ joue, son $n^{ème}$ coup aura une évaluation supérieure ou égale au seuil.

6. Le solveur QuaCode et l'état de l'art

Nous avons développé QuaCode [3] un solveur QCSP construit au-dessus de la librairie CSP GeCode [4]. Ainsi, toute contrainte présente dans GeCode est disponible dans QuaCode. QuaCode implémente l'algorithme 1 pour chercher une stratégie gagnante.

Nous avons réaliser un ensemble de tests sur des Intel i5, 1.8GHz, 4GB RAM tournant sous Linux. Chaque exécution n'utilise qu'un seul cœur et le temps maximum de calcul est fixé à une demie heure. Dans notre expérimentation, nous comparons QuaCode aux deux autres solvers de l'état de l'art : Queso [5] et Qecode [6]. Queso est un solver de QCSP écrit *from scratch* en Java par P. Nightingale. Queso n'est plus maintenu mais les sources sont toujours disponibles. Qecode est un solver QCSP/QCSP+ basé sur GeCode. Pour résoudre un QCSP+, Qecode construit des problèmes CSP dont toutes les variables sont quantifiées existentiellement. L'algorithme de recherche en profondeur d'abord de GeCode est appelé à chaque fois qu'il faut résoudre un CSP et les résultats sont capturés et analysés par Qecode. Pour résoudre un QCSP+, de nombreux CSP doivent être résolus. La conception du QCSP+ doit être prise soigneusement en compte car Qecode nécessite de connaître l'alternance des quantificateurs pour connecter les pièces du QCSP+. Notons que le nombre de nœuds retourné par Qecode est sous évalué comparé au vrai nombre de nœuds ouverts. En effet, lorsque Qecode fournit à GeCode un CSP dans lequel il reste des variables à instancier et donc des nœuds à ouvrir, ceux-ci ne sont pas comptabilisé dans le résultat affiché. La valeur affichée est donc plus en lien avec ceux des CSP résolus par GeCode que ceux réellement ouverts.

Pour évaluer QuaCode versus l'état de l'art, nous utilisons le *Jeu de Nim*, le *MatrixGame* et le *Puissance Quatre* présentés au paragraphe 3.

---

3. http://quacode.barichard.com
4. http://www.gecode.org
5. http://pn.host.cs.st-andrews.ac.uk/
6. http://www.univ-orleans.fr/lifo/software/qecode/QeCode.html

6.1. Le MatrixGame

Même si le *MatrixGame* est souvent présenté comme un problème QCSP+, il n'y a pas de règle à satisfaire entre chaque tour. Toutes les contraintes dépendent de l'ensemble des variables. Ainsi, ce problème ne réclame aucune des spécificités d'un QCSP+ et peut être modélisé en un QCSP sans perte d'efficacité. Toutes les instances utilisées admettent une stratégie gagnante.

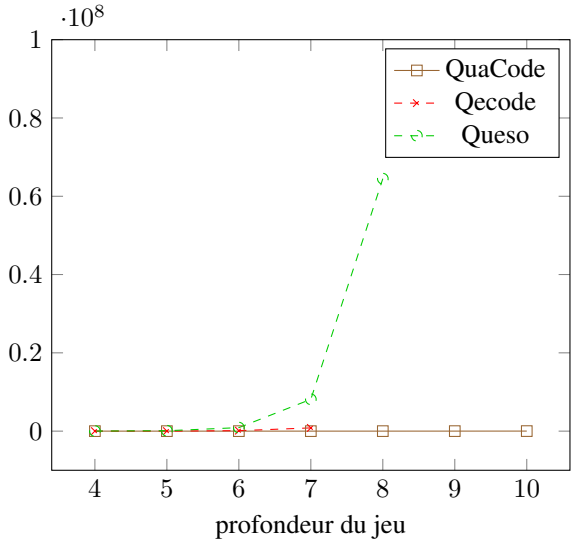


Figure 5. Nombre de propagations pour le MatrixGame

Nous notons que Qecode dépasse le temps maximum accordé après simplement une profondeur du jeu de 7 et Queso de 8. Nous observons une rupture qui apparaît tôt ou tard selon le solveur. Cela est illustré par l'explosion du nombre de propagations de Queso pour une profondeur de 8 (cf. figure 5) et par l'augmentation du nombre de nœuds de Qecode pour une profondeur de 7 (cf. figure 6). Queso et QuaCode étant basés sur des algorithmes de recherche quantifiés similaires, la différence de performance s'explique par la librairie de contraintes sous-jacente. En effet, QuaCode est basé sur GeCode et bénéficie de toutes ses contraintes. Contraintes qui ont été largement éprouvées par la communauté CSP.

6.2. Le Jeu de Nim

Nous abordons maintenant le *Jeu de Nim*. Les figure 7 et 8 reportent respectivement le nombre de propagations et le nombre de nœuds de QuaCode comparés à Queso et Qecode. Nous constatons en particulier que le nombre de propagations et de nœuds de Queso s'accroissent significativement par rapport à ceux de QuaCode et Qecode. De fait, Qecode est conçu pour être efficace sur ce genre de QCSP et ainsi

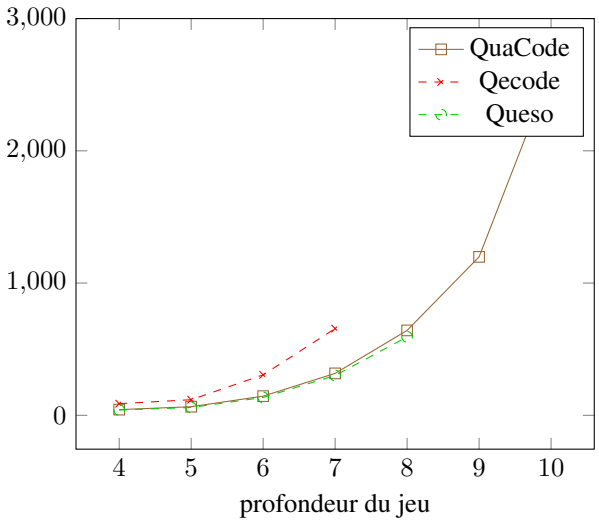


Figure 6. Nombre de nœuds pour le MatrixGame

évite les branches trivialement satisfaites de l'arbre de recherche comme le fait l'outil de *coupure* dans le cas de QuaCode. Les résultats de Qecode et QuaCode sont similaires. Cela montre que l'on peut obtenir la même efficacité qu'un solveur dédié grâce à l'outil *coupure*.

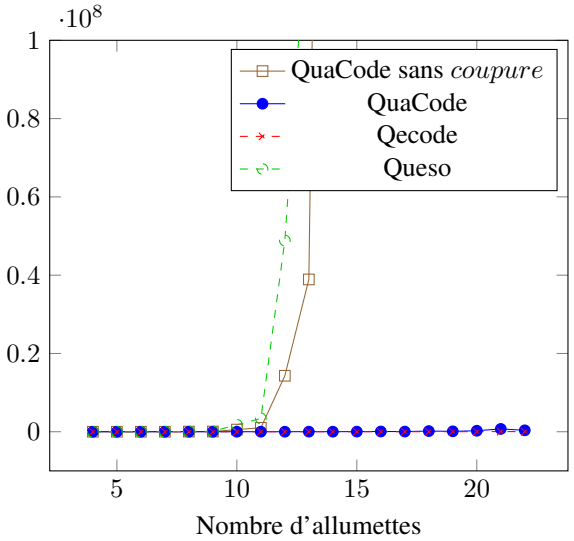


Figure 7. Nombre de propagations pour le jeu de Nim

La figure 9 montre que lorsque le nombre d'allumettes dépasse 12, le temps de calcul de Queso excède alors la demie heure tandis que QuaCode et Qecode nécessitent

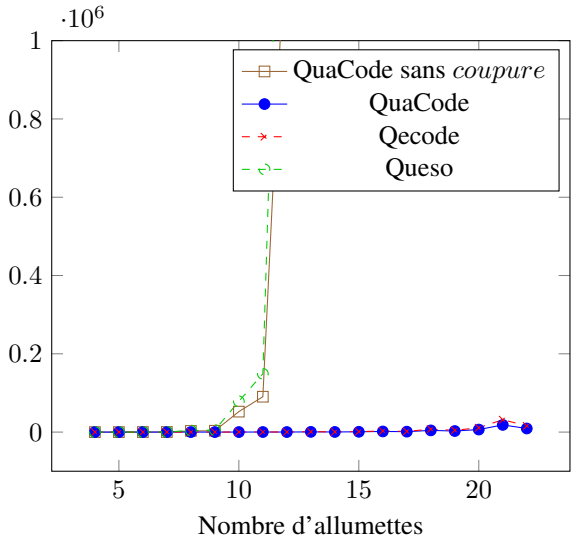


Figure 8. Nombre de nœuds pour le jeu de Nim

moins d'une seconde. QuaCode surperforme Qecode quelques soient les instances. Pour la dernière instance, c'est même deux fois plus rapide. QuaCode et Qecode sont tous les deux basés sur GeCode ainsi les contraintes sont utilisées de la même manière. Mais en Qecode, la résolution des CSP est laissée à GeCode et aucune connaissance ne venant du QCSP ne peut être utilisée pendant cette phase. QuaCode n'apparaît pas comme une *boîte noire* pour la recherche d'un scénario et utilise toutes les connaissances disponibles du problème pour clore au plus vite la recherche. Cela explique le gain d'efficacité de QuaCode sur Qecode.

Ces figures montrent aussi que si l'outil de *coupure* n'est pas mis en œuvre le « talon d'Achille » apparaît comme pour Queso.

6.3. Le Puissance Quatre

Le dernier problème testé est le *Puissance Quatre* déjà étudié par P. Nightingale dans (Nightingale, 2009). Nous utilisons le modèle de base et comparons, selon différents paramètres, Queso et QuaCode. Malheureusement, ce problème ne dispose pas de formulation pour Qecode ; ainsi nous n'utilisons pas Qecode pour ces tests. Dans (Nightingale, 2007), l'auteur adapte la méthode des *valeurs pures*, issue des *littéraux monotones* pour SAT (Bennaceur, 2004), aux QCSP. Ainsi, Queso peut élaguer des valeurs dans les domaines des variables universelles pour éviter des branches inutiles de l'arbre de recherche. Ce mécanisme est puissant mais il est très couteux et doit être utilisé avec prudence. Le *Puissance Quatre* est un classique des jeux à deux joueurs et bénéficie pleinement du théorème 1 portant sur l'utilisation de la coupure. Notre but ici est de comparer un solveur qui utilise l'optimisation basé sur les valeurs

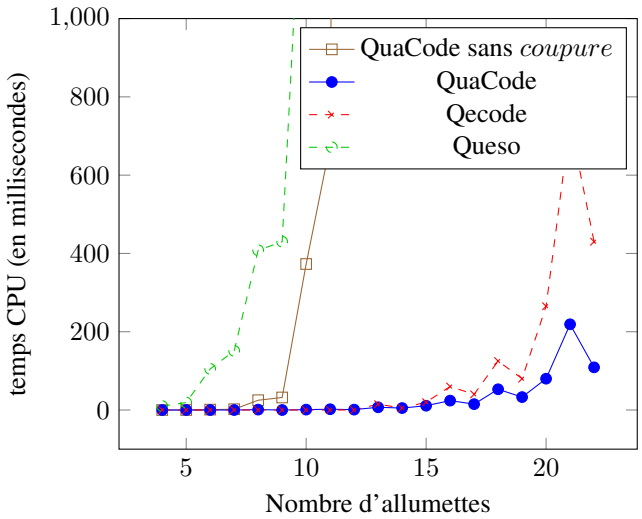


Figure 9. Comparison des temps de calcul pour le jeu de Nim

pures avec un solveur qui exploite le théorème 1 et l'outil de coupure. Notons que la modélisation utilisée du *Puissance Quatre* est favorable à l'utilisation des valeurs pures. En effet, elle ajoute des variables existentielles *ombres* qui facilitent la détection des valeurs pures.

Tableau 1. Comparaison pour le Puissance Quatre *des solveurs QCSP selon différents paramètres (« Opt. » pour « options », "VP" pour « valeurs pures », « QC » pour « QuaCode », « Qo » pour « Queso », « TD » pour « temps dépassé » i.e. plus que 1 800 000 ms)*

Taille du plateau		Propagations, nœuds et temps				
col	ligne	Solveur	Opt.	Propagations	Nœuds	Temps en (ms)
4	4	QC		798 636 092	928 645	142 180
4	4	QC	coupure	6 766 190	7 800	885
4	4	Qo		512 177 815	845 296	55 207
4	4	Qo	VP	20 761 455	25 141	2 813
4	5	QC				TD
4	5	QC	coupure	83 364 319	76 117	11 874
4	5	Qo				TD
4	5	Qo	VP	1 037 360 167	775 784	112 718
5	4	QC				TD
5	4	QC	coupure	940 296 419	979 244	138 768
5	4	Qo				TD
5	4	Qo	VP			TD

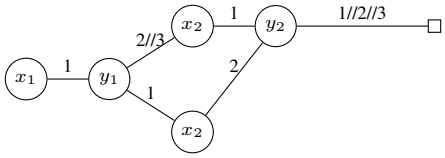
L'ensemble des résultats sont présentés dans le tableau 1. Aucun des solveurs (sans les valeurs pures ou l'outil de coupure) n'est à même de résoudre des instances avec plus de 4 lignes ou colonnes dans le temps imparti. Cela montre que l'élagage fait par les valeurs pures ou l'outil de coupure est indispensable pour résoudre efficacement le problème. De fait, pour chaque instance, le nombre de nœuds explorés et le nombre de propagations pour Queso avec valeurs pures et QuaCode avec l'outil de coupure sont très inférieurs à ceux pour Queso ou QuaCode seuls. Ces résultats se traduisent en une réduction drastique du temps de calcul. La seconde observation est que l'utilisation de l'outil coupure élague plus de nœuds que l'utilisation des valeurs pures. En effet, QuaCode avec l'outil coupure surperforme Queso avec valeurs pures sur toutes les instances. Sur l'ultime instance, l'unique solveur qui la résout dans le temps imparti est QuaCode.

7. Discussion

Dans le cadre des solveurs QCSP, deux points d'efficacité sont à discuter : celui de la représentation de la stratégie et celui des solveurs de type QuaCode dont l'efficacité ne peut pas reposer uniquement sur l'algorithme 1 de recherche.

7.1. Pour une gestion efficace des stratégies

Encore une fois sous les hypothèses classiques de la théorie de la complexité, il y a peu d'espoir de pouvoir encoder d'une manière ou d'une autre toute stratégie, gagnante ou non, sous une forme polynomiale (Coste-Marquis *et al.*, 2006). La représentation classique présentée dans la section 2 est celle d'une arborescente complète qui est évidemment utile pour la définition de la sémantique des QCSP mais inutilisable en pratique (car exponentielle en nombre de nœuds par rapport au nombre de variables universellement quantifiées). De fait, une stratégie peut être optimisée en ne stockant pas un arbre mais un graphe orienté acyclique (ou DAG) qui partage les sous-arbres identiques ; de même les arcs partant d'un nœud étiqueté par un variable universelle et partageant le même sous arbre peuvent aussi être factorisés. Le DAG ci-dessous représente la stratégie gagnante de la figure 2 pour le *Jeu de Nim* :



En fait pour une stratégie calculée par un algorithme de recherche comme celui de l'algorithme 1, le DAG représentant la stratégie restreint aux variables sur lesquelles l'algorithme à brancher associé à l'ensemble des contraintes initiales et à l'algorithme de propagation est suffisant pour décrire toute stratégie. Ainsi le solveur de CSP sous-jacent sert aussi bien à calculer la stratégie *off line* qu'à la parcourir *on line*.

7.2. Pour un solveur QCSP plus efficace

Une des possibilités pour rendre le solveur plus efficace est d'apprendre la structure du problème en stockant des *nogoods*, respectivement des *goods* pour élaguer des parties de l'espace de recherche car redondants en échec, respectivement en succès (Bacchus, Stergiou, 2007). Ces techniques sont nées dans le cadre des CSP (Prosser, 1993) et se sont montrées très performantes dans le cadre des QBF et de la recherche dirigée par les conflits pour SAT (Marques-Silva, K.Sakallah, 1996) et la programmation logique non monotone (Gebser *et al*., 2007). Mais cette option de recherche pour rendre efficace un solveur QCSP demande de modifier le cœur du solveur CSP sous-jacent et se révèle donc difficile à réaliser, car contraire à sa philosophie, dans le cas de QuaCode.

Une autre des possibilités pour rendre les solveurs QCSP plus efficace est de guider leur recherche soit grâce à des heuristiques internes de choix de variable dans le bloc de quantificateurs considéré [7] ou de choix de valeur à explorer (Stynes, Brown, 2009), soit grâce à des heuristiques externes basées sur la résolution en parallèle et coopérative avec des métaheuristiques, type algorithme de Monte Carlo (Kocsis, Szepesvári, 2006). C'est ce dernier choix qui a était fait prioritairement pour permettre à QuaCode d'être un système ouvert. La figure 10 présente l'architecture actuelle de la coopération entre le solveur QCSP complet et l'algorithme stochastique : le solveur QCSP complet basé sur l'algorithme 1 de recherche quantifiée utilisant la bibliothèque CSP GeCode est guidé heuristiquement de manière asynchrone par un algorithme stochastique via un mécanisme intermédiaire de communication que nous nommons « SIBus » (pour *Search Information Bus*).

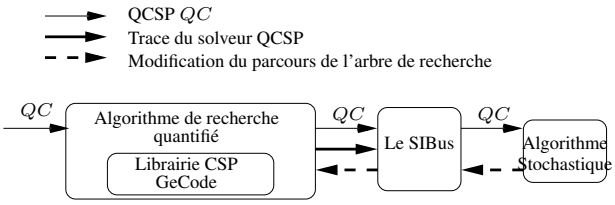


Figure 10. L'architecture actuelle de la coopération entre le solveur complet et l'algorithme stochastique via le SIBus

Si dans le cas des CSP (et de SAT), la vérification que l'instanciation calculée est bien un modèle est polynomiale en temps, il n'en est pas de même pour les QCSP (et les QBF) pour lesquelles le test est co-NP-complet (Kleine Büning, Zhao, 2004). Or l'intérêt des méthodes de type stochastique est de calculer rapidement et en très grand nombre des solutions potentielles. Ainsi WalkQSat (Gent *et al*., 2003), une extension aux QBF de l'algorithme de recherche locale WalkSat (Selman *et al*., 1994),

7. Deux quantificateurs universels ou deux quantificateurs existentiels peuvent être permutés sans mettre en cause la sémantique du QCSP mais il n'en va pas de même en général d'un quantificateur existentiel avec un quantificateur universel. Les heuristiques de choix de variable relèvent donc du choix (dynamique) d'une permutation dans un bloc de quantificateurs identiques.

a montré les limites de l'approche métaheuristique pour le calcul des solutions des QBF et a été abandonnée. Par contre, tester si un scénario est perdant (ou gagnant) est polynomial. Il est alors garanti qu'un tel scénario perdant ne pourra appartenir à aucune stratégie gagnante (tandis qu'un scénario gagnant peut ou peut ne pas appartenir à une stratégie gagnante). L'algorithme stochastique qui sert d'heuristique à l'algorithme complet de recherche quantifié ne calcule donc pas une stratégie au QCSP. Le schéma de fonctionnement général de l'algorithme stochastique en coopération avec l'algorithme complet est donc le suivant : l'algorithme stochastique, par des exécutions polynomiales sur des instances complètes sur le CSP, cherche à identifier des espaces non solutions du CSP sous-jacent. Il réordonne ensuite l'ordre de parcours des valeurs des domaines des variables de manière à retarder l'exploration de ces espaces par le solveur complet. Il guide ainsi le solveur QCSP vers des zones de l'espace de recherche ayant plus de chances de contenir une stratégie gagnante sans modifier la complétude de ce dernier. Le SIBus est libre et disponible avec des exemples de communication sur simple demande.

8. Perspectives et conclusion

Le formalisme QCSP permet de modéliser des problèmes sous-contraintes complexes et dans lesquels certaines variables peuvent être quantifiées universellement. Ce formalisme est bien adapté à la représentation des jeux à deux joueurs à horizon fini. En effet, la richesse du formalisme CSP d'une part, fournit les outils pour modéliser un ensemble d'interactions complexes entre les variables de décisions permettant ainsi de représenter les règles du jeu. La possibilité de quantifier universellement des variables d'autre part, permet de représenter un adversaire à battre. Pour résoudre ces problèmes, nous proposons QuaCode un solveur QCSP, adapté à ce formalisme, construit au-dessus de la bibliothèque CSP GeCode. GeCode fournit sa bibliothèque de contraintes à laquelle ont été ajoutés un algorithme de résolution quantifié et l'outil coupure, permettant de rendre la résolution plus efficace en tirant parti de certaines particularités des QCSP.

Toutefois, beaucoup de jeux comme le *Puissance Quatre*, n'ont pas de stratégie gagnante pour une partie vierge (non commencée). En effet, si tel était le cas, le jeu n'aurait que peu d'intérêt. Malgré tout, des bonnes stratégies existent et parmi celles-ci, il est possible d'en extraire les meilleures. Pour pouvoir modéliser ce problème de manière simple et élégante, il faudrait relâcher le caractère *omniscient* du joueur universel. Nous travaillons actuellement à fournir un cadre légèrement plus souple que les QCSP tout en conservant la simplicité et la richesse du formalisme initial. Nous pensons que cela permettra de modéliser plus naturellement des jeux à deux joueurs réels pour lesquels il n'existe pas de stratégie gagnante initiale.

Bibliographie

Ansotegui C., Gomes C., Selman B. (2005). Achilles' heel of QBF. In *Proceedings of the 20th national conference on artificial intelligence (aaai'05)*, p. 275-281.

Bacchus F., Stergiou K. (2007). Solution directed backjumping for QCSP. In *Proceedings of the 13th international conference on principles and practice of constraint programming (cp'07)*, p. 148-163.

Barichard V., Stéphan I. (2014). The *cut* tool for QCSP. In *Proceedings of the 26th ieee international conference on tools with artificial intelligence (ictai'14)*, p. 883-890.

Barichard V., Stéphan I. (2014). L'outil *coupure* pour les QCSP. In *Actes des dixièmes journées francophones de programmation par contraintes (jfpc'14).*

Benedetti M., Lallouet A., Vautard J. (2007). QCSP made practical by virtue of restricted quantification. In *Proceedings of the 20th international joint conference on artificial intelligence (ijcai'07)*, p. 38-43.

Benedetti M., Lallouet A., Vautard J. (2008). Modeling adversary scheduling with QCSP+. In *Proceedings of the 23th acm symposium on applied computing (sac'08)*, p. 151-155.

Bennaceur H. (2004). A comparison between SAT and CSP techniques. *Constraints*, vol. 9, n° 2, p. 123-138.

Bordeaux L., Cadoli M., Mancini T. (2005). CSP Properties for Quantified Constraints: Definitions and Complexity. In *Proceedings of the 20th national conference on artificial intelligence (aaai'05)*, p. 360-365.

Bordeaux L., Monfroy E. (2002). Beyond NP: Arc-Consistency for Quantified Constraints. In *Proceedings of the 8th international conference on principles and practice of constraint programming (cp'02)*, p. 371-386.

Bordeaux L., Zhang L. (2007). A solver for quantified Boolean and linear constraints. In *Proceedings of the 2007 acm symposium on applied computing (sac '07)*, p. 321-325.

Börner F., Bulatov A., Chen H., Jeavons P., Krokhin A. (2009). The complexity of constraint satisfaction games and QCSP. *Information and Computation*, vol. 207, n° 9, p. 923-944.

Börner F., Bulatov A., Jeavons P., Krokhin A. (2003). Quantified Constraints: Algorithms and Complexity. In *Proceedings of the 17th international workshop on computer science logic (csl'03)*, p. 58-70.

Cadoli M., Giovanardi A., Schaerf M. (1998). An Algorithm to Evaluate Quantified Boolean Formulae. In *Proceedings of the 15th national conference on artificial intelligence (aaai'98)*, p. 262-267.

Chen H., Madelaine F., Martin B. (2015). Quantified Constraints and Containment Problems. *Logical Methods in Computer Science*, vol. 11, n° 3.

Coste-Marquis S., Fargier H., Lang J., Le Berre D., Marquis P. (2006). Representing Policies for Quantified Boolean Formulae. In *Proceedings of the 10th international conference on principles of knowledge representation and reasoning (kr'06)*, p. 286-296.

Gebser M., Kaufmann B., Neumann A., Schaub T. (2007). Conflict-Driven Answer Set Solving. In *Proceedings of the 20th international joint conference on artificial intelligence (ijcai'07)*, p. 386-392.

Gent I., Hoos H., Rowley A., Smyth K. (2003). Unsing Stochastic Local Search to Solve Quantified Boolean Formulae. In *Proceedings of the 9th international conference on principles and practice of constraint programming (cp'03).*

Gent I., Nightingale P., Rowley A. (2004). Encoding Quantified CSPs as Quantified Boolean Formulae. In *Proceedings of the 16th european conference on artificial intelligence (ecai'04)*, p. 176-180.

Gent I., Nightingale P., Rowley A., Stergiou K. (2008). Solving quantified constraint satisfaction problems. *Artificial Intelligence*, vol. 172, n° 6-7, p. 738-771.

Gent I., Nightingale P., Stergiou K. (2005). QCSP-solve: A solver for quantified constraint satisfaction problems. In *Proceedings of 9th international joint conference on artificial intelligence (ijcai'05)*, p. 138-143.

Goultiaeva A., Bacchus F. (2010). Exploiting Circuit Representations in QBF Solving. In *Proceedings of the 13th international conference on theory and applications of satisfiability testing (sat'10)*, p. 333-339.

Kleine Büning H., Zhao X. (2004). On Models for Quantified Boolean Formulas. In *Logic versus approximation, in lecture notes in computer science 3075.*

Kocsis L., Szepesvári C. (2006). Bandit based monte-carlo planning. In *Proceedings of the 17th european conference on machine learning (ecml'06)*, vol. 4212 of LNCS, p. 282-293.

Mamoulis N., Stergiou K. (2004). Algorithms for Quantified Constraint Satisfaction Problems. In *Proceedings of the 10th international conference on principles and practice of constraint programming (cp'04)*, p. 752-756.

Marques-Silva J., K.Sakallah. (1996). GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of ieee/acm international confenrence on computer-aided design (iccad'96)*, p. 220–227.

Nightingale P. (2007). *Consistency and the Quantified Constraint Satisfaction Problem, PhD thesis, University of St Andrews.*

Nightingale P. (2009). Non-binary quantified CSP: algorithms and modelling. *Constraints*, vol. 14, n° 4, p. 539-581.

Papadimitriou C. (1994). *Computational complexity*. Addison-Wesley.

Pralet C., Verfaillie G. (2011). Beyond QCSP for Solving Control Problems. In *Proceedings of the 17th international conference on principles and practice of constraint programming (cp'11)*, p. 744-758.

Prosser P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, vol. 9, p. 268-299.

Sabharwal A., Ansótegui C., Gomes C., Hart J., Selman B. (2006). QBF Modeling: Exploiting Player Symmetry for Simplicity and Efficiency. In *Proceedings of the 9th international conference on theory and applications of satisfiability testing (sat'06)*, p. 382-395.

Selman B., Kautz H., Cohen B. (1994). Noise strategies for improving local search. In *Proceedings of the 12th national conference on artificial intelligence (aaai'94)*, p. 337-343.

Stockmeyer L., Meyer A. (1973). Word problems requiring exponential time. In *Proceedings of the 5th annual acm symposium on theory of computing (stoc '73)*, p. 1-9.

Stynes D., Brown K. (2009). Value ordering for quantified CSPs. *Constraints*, vol. 14, n° 1, p. 16-37.

Tsang E. (1993). Foundations of constraint satisfaction. *Academic Press, London.*

Verger G., Bessiere C. (2006). BlockSolve: a Bottom-Up Approach for Solving Quantified CSPs. In *Proceedings of the 12th international conference on principles and practice of constraint programming (cp'06)*, p. 635-649.

Verger G., Bessiere C. (2008). Guiding Search in QCSP$^+$ with Back-Propagation. In *Proceedings of the 14th international conference on principles and practice of constraint programming (cp'08)*, p. 175-189.