Résolution de SCSP avec borne de confiance pour les jeux de stratégie

Frédéric Koriche, Sylvain Lagrue, Éric Piette, Sébastien Tabary

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

Résumé

Cet article examine un fragment du problème de satisfaction de contraintes stochastiques, représentant les jeux à informations complètes et incertaines. Nous proposons un algorithme de résolution permettant de trouver des stratégies gagnantes pour cette classe de jeux. L'intérêt de ce travail est de permettre la résolution efficace d'un SCSP initial par la résolution successive d'une séquence de « micro-SCSP » à chaque étape du jeu. L'algorithme MAC, proposé pour résoudre les premiers micro-SCSP de la séquence, est couplé à une heuristique de type UCB utilisée pour estimer les valeurs des stratégies sur toute la séquence. Notre approche, MAC-UCB, est validée par une série d'expérimentations sur un ensemble de jeux très divers, décrits en GDL (Game Description Language), et comparée à l'algorithme UCT, qui est la référence actuelle pour cette classe de jeux.

1 Introduction

L'objectif du *General Game Playing* (GGP) est de concevoir des algorithmes de décision, 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. Une compétition est d'ailleurs organisée chaque année par AAAI [7], au cours de laquelle de nombreux algorithmes de jeux s'y confrontent. Les jeux sont décrits dans un langage de représentation, appelé GDL pour *Game Description Language* [10] ; la dernière version de ce langage, GDLII, permet de modéliser des jeux à information incertaine et incomplète [20]. Parmi les algorithmes génériques de jeu, on compte notamment des méthodes de type Monte Carlo [6], la construction automatique de fonctions d'évaluation [4], ou encore la programmation logique [19] et ASP [13]. Bien entendu, le problème GGP n'est pas cantonné aux « univers ludiques » : il permet de modéliser des problèmes de prise de décision séquentielle mono ou multi-agents.

Par son approche déclarative de la résolution de problèmes combinatoires, la programmation par contraintes offre un cadre prometteur pour relever le défi du GGP. Parmi les différents formalismes à base de contraintes proposés pour représenter et résoudre des jeux, on peut citer les QCSP [8], les *strategic* CSP [3] ou encore les *constraint games* [15]. Cependant, la plupart de ces formalismes sont restreints aux jeux à information complète et certaine : à chaque instant, les joueurs ont une information complète de l'état du jeu et leurs actions sont déterministes. Cette étude se focalise sur le formalisme des réseaux de contraintes stochastiques (SCSP) [21], utilisés pour la prise de décision séquentielle, dont l'effet incertain des actions est modélisé par une distribution de probabilités.

Nous examinons un fragment de SCSP, proposé dans [9], permettant de modéliser des jeux GDL à information complète mais incertaine. De manière importante, le SCSP associé à cette classe de jeux peut être décomposé

en une séquence de « micro-SCSP ». A partir de cette décomposition, nous proposons un algorithme de décision séquentielle couplant la méthode MAC (*Maintaining Arc Consistency*) pour résoudre les micro-SCSP, et la méthode des bandits stochastiques UCB (*Upper Confidence Bound*) pour estimer l'utilité des stratégies. Notre algorithme, MAC−UCB, évalué sur un grand nombre d'expérimentations ($\sim 10^5$) réalisées sur des jeux GDL très divers, se révèle être, dans la plupart des scénarios, meilleur que l'algorithme UCT (*Upper Confidence Tree*), qui demeure la référence dans le domaine des jeux de stratégie à information incertaine.

L'article est organisé de la manière suivante. Le cadre SCSP pour les jeux GDL est introduit en Section 2, et l'algorithme MAC−UCB est détaillé en Section 3. Avant de conclure, la Section 4 présente sur une série de jeux GDL avec des temps de réflexion différents, les résultats obtenus par MAC−UCB représentant un joueur, face à UCT représentant le joueur adverse.

2 Un fragment de SCSP pour GDL

Les SCSP examinés dans cette étude étendent le modèle originel de Walsh ([21]) pour traiter les contraintes valuées.

SCSP. De manière formelle, un *Stochastic Constraint Satisfaction Problem* (SCSP) est un tuple $\langle X, Y, D, P, \mathcal{C}, \theta \rangle$ tel que X est un ensemble ordonné de n variables et Y est le sous-ensemble de X spécifiant les variables stochastiques ; D représente l'ensemble des domaines associés aux variables de X, P est l'ensemble des distributions de probabilité appliquées aux domaines des variables stochastiques, $\mathcal{C}$ est l'ensemble des contraintes, et θ est la valeur de seuil comprise entre 0 et 1.

Les variables incluses dans $X \setminus Y$ sont appelées variables de décision et ont la même signification que celles définies dans un CSP classique. Nous notons $D(z)$ le domaine associé à une variable $z \in X$. Plus généralement, pour un sous-ensemble $Z = \{z_1, \cdots, z_k\} \subseteq X$, nous notons $D(Z)$ l'ensemble des tuples de valeurs $D(z_1) \times \cdots \times D(z_k)$.

Chaque contrainte d'un SCSP est une paire $c = \langle scp_c, val_c \rangle$, où scp_c est le sous-ensemble de X, dénommé *portée* de c, et val_c est une fonction qui associe pour chaque tuple $\boldsymbol{\tau} \in D(scp_c)$ une valeur (ou utilité) $val_c(\boldsymbol{\tau}) \in [0, 1] \cup \{-\infty\}$. Une contrainte de décision est une contrainte $c \in C$ où scp_c est restreint aux variables de décisions. À la différence d'une contrainte stochastique dont scp_c porte au moins sur une variable stochastique. Une contrainte $c \in C$ est *dure* si val_c retourne $-\infty$ si c est violée sinon 0 et *souple* si la valeur de retour de la fonction val_c est comprise entre 0 et 1. Si c est une contrainte dure, alors elle peut être représentée de manière usuelle par une relation, notée rel_c, qui contient l'ensemble des tuples autorisés pour scp_c (i.e. les tuples $\boldsymbol{\tau}$ pour lesquels $val_c(\boldsymbol{\tau}) \neq -\infty$).

Étant donné un ensemble de variables $Z = \{z_1, \ldots, z_k\} \subseteq X$, une *instanciation* I de Z est un ensemble de paires variable-valeur $\{(z_1, v_1), \ldots, (z_k, v_k)\}$, tel que $v_i \in D(z_i)$ pour chaque $z_i \in Z$. Une instanciation est dite *complète* si $Z = X$. Pour un sous-ensemble Z' de Z, nous notons $I_{|Z'}$ la projection de I sur Z'. L'utilité de I est donnée par :

$$val(I) = \sum_{c \in \mathcal{C}} val_c(I_{|scp_c})$$

Une politique π est un ensemble d'instanciations complètes, structurées en arbre, dont les nœuds sont les variables de X, et dont les arêtes sont étiquetées par la valeur assignée au nœud parent. Les nœuds associés aux variables de décision ont un seul fils, tandis que les nœuds associés aux variables stochastiques ont un fils pour chaque valeur possible de leurs domaines. Ainsi, chaque instanciation complète de π est identifiée par un chemin depuis la racine de π jusqu'à une feuille. Les feuilles sont étiquetées par l'utilité du chemin correspondant. L'utilité espérée d'une politique π est définie par la somme des utilités des feuilles pondérées par leurs probabilités.

Une *solution* du SCSP est une politique π dont l'utilité espérée est plus grande ou égale au seuil θ. Comme $\theta > -\infty$, les contraintes dures sont nécessairement satisfaites par une solution. Ainsi, une politique π satisfait le SCSP si toutes les contraintes dures sont satisfaites et si l'utilité espérée excède le seuil θ.

Exemple 1 *Considérons le SCSP défini par*

les variables de décision x_1 et x_2 et la variable stochastique y ; les trois variables sont définies sur le domaine $\{0, 1, 2\}$, et la distribution de y est uniforme ($P(0) = P(1) = P(2) = 1/3$). Le réseau possède deux contraintes :

$$c_h(x_1, x_2) = \begin{cases} 0 & si\ x_1 = x_2 \\ -\infty & sinon \end{cases}$$

$$c_s(x_1, y) = \begin{cases} 1 & si\ x_1 + y \geq 1 \\ 0 & sinon \end{cases}$$

Enfin, le seuil θ est fixé à $1/2$. La politique π représentée par la figure 1 est une solution du SCSP, puisqu'elle ne viole pas la contrainte dure c_h et satisfait la contrainte valuée c_s avec une utilité espérée de $2/3 \geq \theta$.

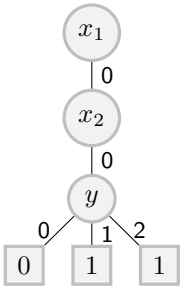


FIGURE 1 – Une politique π

Le problème consistant à trouver une politique solution d'un SCSP est, en général, PSPACE-complet. Il existe cependant une sous-classe intéressante de SCSP pour laquelle le problème de satisfaction est de complexité moindre ($\mathsf{NP}^{\mathsf{PP}}$). Il s'agit des *one stage* SCSP [21], ou micro-SCSP, que l'on note ici μSCSP. Formellement, un μSCSP est un SCSP $(X, Y, D, P, \mathcal{C}, \theta)$, dans lequel la restriction suivante est imposée sur les politiques : un arbre d'instanciations complètes π est une politique de μSCSP si toutes les variables de décision X sont assignées *avant* les variables stochastiques Y. Comme le montre Walsh, tout SCSP défini sur n variables de décision peut être vu comme une séquence de n μSCSP (les contraintes peuvent néanmoins porter sur plusieurs μSCSP).

SCSP d'un programme GDL. Dans [9], un formalisme a été proposé pour traduire des jeux GDL [10] vers un SCSP. Un programme GDL est un ensemble de règles en logique du premier ordre ; pour un horizon T donné, ce programme peut être instancié en T ensembles de règles, définis aux tours $t \in \{1, \cdots, T\}$.

Nous présentons ici de manière informelle le processus de transformation d'un programme GDL à horizon T en SCSP. Pour chaque tour de jeu t, l'ensemble des règles instanciées à t est représenté par un μSCSP_t défini sur l'ensemble des variables $X_t \cup X_{t+1} \cup \{y_t\}$. Deux variables de décision $\mathsf{terminal}_t$ et $\mathsf{control}_t$ sont utilisées pour capturer l'état terminal du jeu et indiquer quel joueur à la main. Chaque fluent apparaissant à l'instant t est associé à deux variables de décision, correspondant aux états du fluent à l'instant t et à l'instant $t + 1$. De la même manière, chaque action apparaissant à t est associée à une variable de décision. Les domaines de ces variables représentent toutes les combinaisons possibles de constantes qui sont instanciées par les atomes du programme GDL. Enfin, chaque règle est associée à une contrainte dont la portée est extraite de façon similaire aux domaines des variables pour identifier toutes les combinaisons de constantes autorisées. En substance, les contraintes représentent les actions autorisées à l'instant t et les transitions possibles entre t et $t + 1$.

Le comportement incertain de l'environnement (appelé random dans GDL) est capturé par la variable stochastique y_t (ex : résultat d'un jet de dés à l'instant t). Le domaine de y_t est donné par les actions possibles de l'environnement à l'instant t et, selon les spécifications GDLII [20], la distribution de probabilités $P(y_t)$ est définie comme uniforme sur son domaine de valeurs [1].

Par construction, le SCSP associé à un programme GDL à horizon T est défini comme une séquence de micro-SCSP $\langle \mu\text{SCSP}_1, \cdots, \mu\text{SCSP}_T \rangle$, dans laquelle les contraintes portant sur les coups légaux et les transitions d'états sont spécifiées pour chaque μSCSP_t, $t \in \{1, \cdots, T-1\}$. Le dernier

1. Néanmoins, et comme précisé dans [20], il est possible de représenter des distributions non-uniformes d'actions possibles en autorisant plusieurs valeurs associées à un choix d'action dans le domaine de y_t.

micro-SCSP de la séquence (μSCSP$_T$), défini sur les variables $X_T \cup \{y_T\}$, joue un rôle particulier : dans un jeu GDL, le score n'est donné qu'à l'horizon T et donc seul μSCSP$_T$ possède des contraintes valuées capturant la fonction de score (dans $[0, 1]$) sur l'état terminal. Cette structuration particulière du SCSP incite naturellement à résoudre le problème de manière séquentielle en explorant itérativement chaque μSCSP$_t$.

Le seuil θ peut être ajusté selon la stratégie désirée : en partant d'un seuil de 0 qui autorise toute politique « faisable » (ne violant aucune contrainte dure), il est possible de construire des SCSP de plus en plus contraints, dont la résolution pour un seuil θ peut tirer parti de la résolution pour un seuil $\theta' < \theta$. En particulier, les solutions des μSCSP$_t$ aboutissant à des stratégies non-optimales pour θ' peuvent être élaguées lors de la résolution du problème au seuil θ.

3 MAC-UCB

Dans cette section, nous présentons notre algorithme de résolution MAC-UCB. A partir d'un SCSP décomposé en une séquence $\langle \mu\text{SCSP}_t \rangle$ de micro-problèmes de résolution de contraintes stochastiques, MAC-UCB cherche une politique solution en maintenant l'arc consistance sur chaque μSCSP$_t$ et en échantillonnant avec borne de confiance l'utilité des solutions du μSCSP$_t$.

Prétraitement. Avant de résoudre un μSCSP, nous utilisons des techniques de prétraitement pour améliorer l'efficacité de résolution. Nous commençons par fusionner les contraintes dures de même portée. Étant donné un μSCSP P, deux contraintes dures c_i et c_j de P telle que $scp_{c_i} = scp_{c_j}$ deviennent une unique contrainte c_k où $rel_{c_k} = rel_{c_i} \cap rel_{c_j}$ et $scp_{c_k} = scp_{c_j} = scp_{c_i}$. Nous supprimons aussi toutes les contraintes unaires (e.g. contraintes c dont $|scp_c| = 1$). Le domaine de la variable impliquée par la contrainte unaire est restreint aux valeurs autorisées par les tuples de la relation associée.

Ensuite, nous identifions les variables universelles dans chaque contrainte. Une variable est universelle sur c si quelque soit la valeur assignée, c est toujours satisfaite. Autrement dit, une variable $x \in scp_c$ est universelle si $|rel_c|$ est égal au produit de la taille du domaine de x avec le nombre de tuples de la relation associée à la contrainte c_i, telle que $scp_{c_i} = scp_c \setminus \{x\}$. Ces variables sont supprimées de la portée des contraintes.

La dernière technique de prétraitement est d'établir la propriété de singleton arc consistance (*Singleton Arc Consistency* noté SAC [5]) sur le réseau de contraintes associé au μSCSP correspondant. Un réseau de contraintes P est singleton arc-consistant si chaque valeur de P est singleton arc-consistante. Une valeur est singleton arc-consistante si une fois assignée à sa variable, elle ne produit pas un réseau arc inconsistant. Nous appliquons cette propriété pour supprimer les valeurs inconsistantes des domaines des variables.

MAC. Une fois le micro-SCSP prétraité, le problème principal est d'énumérer les solutions faisables de ce problème, dont certaines peuvent aboutir à une solution optimale. A notre connaissance, la meilleure méthode pour résoudre un μSCSP est *Forward Checking* (FC) présenté dans [2]. Malheureusement pour un μSCSP contenant un grand nombre de contraintes, FC n'est pas suffisament efficace, à cause de sa faible capacité de filtrage. L'idée de notre approche est de résoudre le μSCSP avec l'algorithme de maintien d'arc consistance (MAC) [16, 17]. Pour rappel, l'algorithme MAC alterne inférence et recherche. Il effectue une recherche en profondeur d'abord avec retour arrière et maintient à chaque nœud une consistance locale : *la consistance d'arc* (AC) [12, 11]. Pour appliquer cet algorithme sur un problème de contraintes stochastiques, nous avons besoin de partitionner les contraintes : d'une part les contraintes de décision (CSP P') et d'autre part les contraintes stochastiques (μSCSP P''). L'algorithme MAC est appliqué sur un CSP classique composé uniquement des contraintes de P'. Les solutions du μSCSP sont identifiées en combinant la résolution du CSP P' avec celle du μSCSP P''.

Premièrement, nous commençons par résoudre le plus petit problème, le μSCSP P''. En effet, pour des jeux GDL, P'' contient seulement

une contrainte sur l'action de l'environnement et une contrainte pour chaque modification de fluent connecté aux actions de l'environnement. Cette partie du problème peut être traitée par une variante de *Forward Checking* (FC) [2] adaptée à notre problème. Comme notre algorithme est utilisé pour la décision séquentielle, notre version énumère toutes les politiques faisables de P'' alors que l'algorithme d'origine ne retourne que la valeur de satisfaction obtenue. L'ensemble des politiques solutions obtenues avec FC sur P'' est exprimable par une contrainte c_s dont la portée est l'ensemble des variables de décisions et la relation possède les valeurs des stratégies gagnantes. Pour réaliser la jointure des deux ensembles P' et P'' (projeté sur les variables de P') nous ajoutons à P' la contrainte c_s. De cette manière, quand nous utilisons l'algorithme classique MAC sur P', il retourne l'ensemble des solutions de P' qui seront les mêmes que celle du μSCSP d'origine. Cette partie de l'arbre de recherche est filtrée par la propriété d'arc consistance pour énumérer rapidement les solutions de P'.

Exemple 2 *Afin d'illustrer la résolution d'un μSCSP P, considérons le μSCSP d'origine donné par les variables de décision $\{x, z\}$ et la variable stochastique y. Le domaine de x est $\{1, 2, 3\}$, et le domaine de y et z est $\{0, 1, 2\}$. La distribution P de y est uniforme sur son domaine. Le réseau possède trois contraintes données par :*

$$c_1(x, y) : x + y > 1$$
$$c_2(y, z) : y + z > 1$$
$$c_3(x, z) : x = z$$

Enfin le seuil θ est fixé à $7/10$.

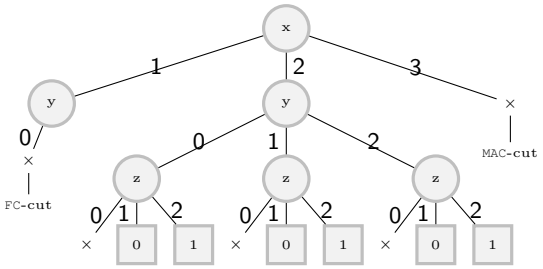
Après partitionnement, la partie CSP P' est donnée par les deux variables de décision x et z et la contrainte c_3. La partie μSCSP P'' est donnée par les trois variables x, y, z et les contraintes c_1 et c_2.

L'algorithme FC retourne pour le problème stochastique les politiques $(x = 2\,;\, z = 2)$ et $(x = 3\,;\, z = 2)$. La satisfaction associée pour ces politiques est de 1. Ainsi la contrainte c_s est ajoutée à P' est :

$$c_s(x, z) : x = z = 2 \mid x = z = 3$$

L'algorithme MAC retourne pour ce problème la solution : $(x = 2\,;\, z = 2)$.

La figure ci-dessous illustre l'arbre de recherche obtenu quand nous résolvons ces deux problèmes.



UCB. La résolution des différents μSCSP correspondant aux états d'un jeu GDL n'est pas suffisante. Comme nous l'avons vu précédemment dans la section 2, seul le dernier μSCSP de la séquence (μSCSP_T) possède des contraintes valuées et retourne une utilité comprise dans $[0, 1]$. C'est pourquoi après chaque résolution de μSCSP_t, nous avons besoin de simuler les états suivants afin d'estimer l'utilité des actions accomplies jusqu'à t. Dans cet objectif, nous utilisons l'algorithme standard des bandits stochastiques *Upper Confidence Bound* (UCB) [1] qui échantillonne les états suivants de $t+1$ jusqu'à T, et retourne une borne de confiance sur l'utilité des actions accomplies jusqu'à t. Rappelons qu'UCB commence par jouer chaque action possible une fois. Puis, il choisit une action i qu'il maximise par $\bar{x}_i + \sqrt{(2 \ln n)/n_i}$, où $\bar{x}_i$ est l'espérance empirique des utilités obtenues à partir de l'action i, n_i est le nombre de fois où l'action i a été choisie jusqu'ici, et n est le nombre total de simulations. Pour nos problèmes, $10,000$ simulations sont réalisées et UCB calcule une utilité avec ces données. Finalement, nous continuons la résolution du problème en instanciant le prochain μSCSP avec les valeurs obtenues sur la politique possédant l'utilité la plus grande, afin d'explorer l'arbre de recherche optimalement.

Élagage. Rappelons que la tâche de décision séquentielle associée à un jeu de stratégie est un problème d'optimisation. De manière classique, ce problème peut être géré par une séquence de

problèmes de satisfaction (SCSP) dont le seuil est augmenté progressivement.

Le seuillage progressif du problème d'optimisation est exploité dans MAC−UCB par deux méthodes d'élagage. La première est probabiliste et tire parti de la borne de confiance d'UCB ; rappelons que la qualité de chaque solution du problème P associé à un μSCSP est estimée par échantillonnage. Lorsqu'un nombre suffisant de valeurs de la variable stochastique se situent en dessous de la borne de confiance, l'instanciation des variables de décision, couplée avec les valeurs de la variable stochastique, est supprimée des solutions du sous-problème stochastique P'' de P. Par répercussion, le nombre de tuples de la contrainte c_s associée aux solutions de P'' projetées sur les variables de décision de P est réduit, ce qui facilite la résolution par MAC du problème déterministe P'. En d'autres termes, plus le seuil θ augmente, plus rapide est la résolution du μSCSP.

La deuxième forme d'élagage est déterministe, et consiste simplement à ajouter à UCB un cache lui permettant de connaître les feuilles déjà explorées [2]. En combinant cette mise en cache avec la connaissance du nombre de coups légaux à chaque état, il est possible de déduire si tout le sous-arbre après cet état a déjà été exploré ou non. Ainsi lorsqu'une solution d'un μSCSP donne lieu à un sous-arbre dont la valeur espérée est en dessous de θ, cette solution peut être éliminée avec certitude. En augmentant itérativement le seuil θ, UCB peut exploiter son cache pour éliminer dans chaque μSCSP les solutions sous-optimales pour θ.

4 Résultats expérimentaux

Après un tour d'horizon des SCSP et de MAC−UCB, nous sommes à présent en mesure de présenter les résultats expérimentaux obtenus sur divers jeux avec différents délais de décision pour les joueurs MAC−UCB et UCT.

Benchmarks. Afin de répondre à l'objectif du *General Game Playing* (GGP) [14], nous avons choisi 13 jeux très divers, décrits en GDLII, qui varient selon le type d'environnement (déterministe vs. stochastique), le nombre d'actions et de fluents, le nombre et la taille des contraintes induites par les règles et, naturellement, la fonction objectif (contrainte souple) associée au but.

— *Awale* est un jeu de stratégie à 2 joueurs avec 48 graines et 2 rangées de 6 trous. Chaque joueur contrôle les 6 trous de son coté du plateau. Le jeu commence avec 4 graines dans chaque trou. Le but du jeu est de capturer plus de graines que son opposant. A chaque tour, un joueur choisit un des 6 trous sous son contrôle et distribue toutes les graines présentes dans les trous suivants.

— *Backgammon* est un célèbre jeu de plateau à 2 joueurs avec 2 dés et 15 pièces par joueur. Les pièces sont déplacées selon un lancé de dés et un joueur gagne quand il supprime toutes les pièces de son plateau.

— *Bomberman* est un jeu de labyrinthe très populaire dans l'univers vidéo-ludique. Le but est de placer des bombes pour tuer tous les ennemis et détruire les obstacles.

— *Can't stop* est un jeu de plateau avec 4 dés et 3 moines par joueur, notre version du jeu impliquant seulement 2 joueurs. Le plateau inclut 9 montées de différentes tailles, le but étant d'atteindre le haut des 3 montées choisies par les moines.

— *Checkers* est le célèbre jeu de dames, implémenté ici sur un damier 8×8, avec 2 joueurs contrôlant leurs pions, le but étant d'éliminer ou bloquer les pions de l'adversaire.

— *Chess* est l'incontournable jeu d'échecs sur échiquier de 64 cases, avec 2 joueurs contrôlant 16 pièces, le but étant de mettre « échec et mat » le roi adverse, situation de prise sans qu'aucune action puisse y remédier.

— *Kaseklau* est un petit jeu de plateau à 2 joueurs avec un chat et une souris. Le but est de lancer les 2 dés pour avancer le chat et la souris sur différentes cases en ramassant les fromages placés par dessus quand le chat n'attrape pas la souris.

— *Orchard* ou « verger » est un jeu coopératif de plateau à 2 joueurs utilisant un dé à 6 faces. A chaque tour, si le dé tombe sur un des 4 arbres, le joueur qui a la main enlève un

2. Dans nos expérimentations, 32 GB ont été alloués pour le cache et cette limite n'a jamais été atteint.

fruit de l'arbre, si le dé tombe sur « panier », le joueur prend deux fruits de son choix, et si le dé tombe sur « corbeau », alors ce dernier perd une pièce. Le but du jeu est de récupérer collectivement tous les fruits avant que le corbeau perde toutes ses pièces.

— *Othello* est encore un jeu populaire de plateau 8×8 à 2 joueurs disposant de 64 pions bicolores. Le jeu s'arrête lorsqu'un aucun joueur ne peut plus placer de pion sur le plateau, le gagnant étant celui qui a placé le plus de pions.

— *Pacman* est un célèbre jeu d'arcade où le joueur contrôle *Pac-Man* au travers d'un labyrinthe mangeant des points et des fruits et évitant 4 fantômes souhaitant attraper le personnage. Le joueur gagne la partie s'il mange tous les points sans se faire attraper.

— *Pickomino* est un jeu de 8 dés et de 16 tuiles incluant 1 à 4 vers. A chaque tour, le joueur obtient un score issu du lancé de dés correspondant à la valeur d'une tuile. Le but est d'obtenir le maximum de vers en récupérant les tuiles contenant le plus de vers.

— *Tic-tac-toe* est un jeu déterministe bien connu avec 2 joueurs (X et O) qui marquent une grille de taille 3×3.

— *Yahtzee* est un jeu à 2 joueurs où le but est d'obtenir le plus grand score en lançant 5 dés. À chaque tour, les dés sont lancés jusqu'à 3 fois afin d'obtenir l'une des 13 combinaisons, chacune étant associée à un score.

Le tableau 1 présente les paramètres du μSCSP correspondant à la traduction de ces jeux et le temps nécessaire pour l'obtenir. Nous indiquons le nombre de variables, la taille maximum des domaines, le nombre de contraintes et le temps de traduction en secondes. Le jeu le plus difficile à traduire est *Backgammon* possédant un grand nombre de variables, chacune avec un domaine important et un nombre important de contraintes de grande portée. *Awale*, *Can't Stop*, *Chess*, *Pickomino* et le *Yahtzee* sont aussi intéressants de part la taille du domaine ou le nombre de contraintes.

Protocole. Dans le but de confronter notre algorithme à UCT (*Upper Confidence Tree*), nous avons implémenté un version de ce dernier, fondée sur l'analyse des jeux multi-joueurs [18]. Par souci d'équité, nous avons aussi ajouté un cache à UCT, lui permettant tout comme MAC−UCB de connaître par avance les sous-arbres qui ont déjà été explorés. Nous avons réalisé 390,000 instances de « matchs » entre UCT et MAC−UCB sur l'ensemble des jeux. Pour chaque jeu, un joueur suit la stratégie d'UCT et l'autre joueur celle de MAC−UCB. 5,000 instances de jeux sont réalisées sur différents temps (10s, 20s, 30s, 40s, 50s, 60s) par action.

Jeu	#vars	maxDom	#const	temps
Awale	19	37	73	94
Backgammon	76	768	86	347
Bomberman	145	64	42	31
Can't Stop	16	1,296	409	248
Checkers	86	262,144	52	43
Chess	71	4,096	50	76
Kaseklau	18	7,776	106	35
Orchard	9	146,410	40	12
Othello	81	65	29	51
Pacman	93	64	22	36
Pickomino	29	1,679,616	223	172
TicTacToe	19	18	37	0
Yahtzee	12	30	8,862	182

TABLE 1 – les jeux traduits en μSCSP et leurs caractéristiques.

Résultats. Le tableau 2 présente le pourcentage de victoires pour ces matchs sur chaque jeu avec 30 secondes par action. Nous indiquons aussi l'écart type (σ) correspondant au pourcentage de victoires de MAC−UCB. Pour tous les jeux, il apparaît que MAC−UCB est en moyenne plus performant qu'UCT, le phénomène s'accentuant en fait pour des délais plus grands. Les résultats sont particulièrement marquants sur les jeux stochastiques, tels que *Bomberman*, *Can't Stop*, *Kaseklau*, *Pacman*, *Pickomino*, *Yahtzee*, et surtout le *Backgammon*, pour lequel MAC−UCB gagne dans environ 70 % des cas. Pour jeux déterministes, l'écart est moins important (excepté *Othello*), ce qui s'explique par le fait que, sans action stochastique, MAC−UCB ne peut pas bénéficier du filtrage probabiliste. Enfin, pour le jeu coopératif *Orchard*, les deux algorithmes atteignent une moyenne de 70 %, Il est connu que ce pourcentage est le maximum possible pour la stratégie optimale.

Jeu	UCT	MAC-UCB	σ
Awale	43.3 %	56.7 %	1.63 %
Backgammon	31.8 %	68.2 %	5.49 %
Bomberman	34.6 %	65.4 %	6.32 %
Can't Stop	28.3 %	71.7 %	6.43 %
Checkers	38.8 %	61.2 %	2.12 %
Chess	46.2 %	53.8 %	1.75 %
Kaseklau	28.6 %	71.4 %	6.56 %
Orchard [3]	70.2 %	70.2 %	3.41 %
Othello	20.7 %	79.3 %	1.41 %
Pacman	30.9 %	69.1 %	2.73 %
Pickomino	36.6 %	63.4 %	4.89 %
Tictactoe	34.3 %	65.7 %	0.89 %
Yahtzee	28.8 %	71.2 %	5.48 %

TABLE 2 – Résultats pour les différents jeux GDL avec 30 secondes par coup

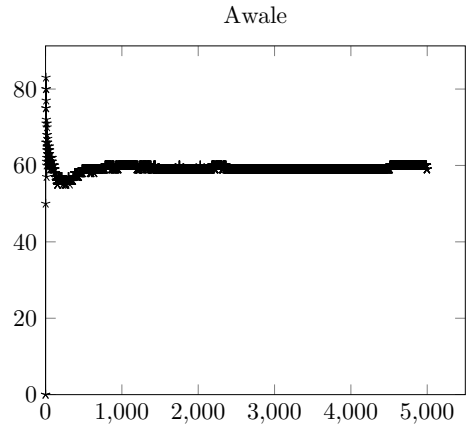


FIGURE 2 – En abscisse, le nombre d'instance de jeux et en ordonnée, le pourcentage de victoires de MAC-UCB contre UCT avec 30 secondes par coup sur *Awale*.

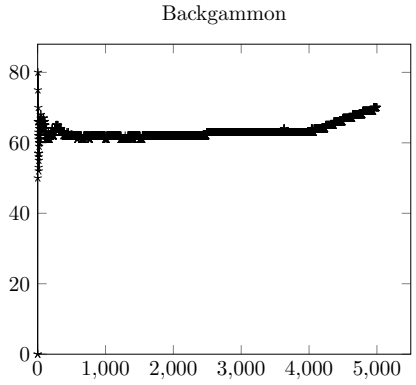


FIGURE 3 – En abscisse, le nombre d'instance de jeux en ordonnée, le pourcentage de victoires de MAC-UCB contre UCT avec 30 secondes par coup sur *Backgammon*.

Les figures 2 et 3 présentent, respectivement pour *Awale* et *Backgammon*, l'évolution du pourcentage de victoires de MAC-UCB contre UCT avec 30 secondes par coup, en faisant évoluer le nombre d'instances jusqu'à 5,000. Pour le jeu déterministe *Awale*, la performance est pratiquement stationnaire au bout de 1,000 instances. Ici, seul le filtrage réalisé par MAC sur la partie CSP des réseaux stochastiques a un effet sur cette performance. A l'opposé, pour le jeu stochastique *Backgammon*, la performance de MAC-UCB s'améliore sensiblement, avec une augmentation de victoires de 10% au bout de 4,000 parties. Cette amélioration peut s'expliquer par l'augmentation du nombre d'élagages probabilistes et déterministes induits par UCB, dont le cache grossit progressivement. Un phénomène similaire est observé pour les autres jeux stochastiques.

La figure 4 montre le pourcentage de victoires de MAC-UCB contre UCT sur le jeu *Can't Stop*, avec différents temps par action (10 à 60 secondes). L'écart de performances entre MAC-UCB et UCT augmente sensiblement avec le temps de décision autorisé ; MAC-UCB obtient 59.9 % de victoires en 10 secondes par coup, et 78.9 % de victoires pour 60 secondes par coup. Cet écart est expliqué par la capacité de MAC à résoudre un nombre de plus en plus grand de μSCSP dans la séquence, ce qui facilite le travail d'exploration d'UCB.

Nous concluons cette partie expérimentale en mettant l'accent sur le dilemme de MAC-UCB entre la résolution (partie MAC) et l'exploration (partie UCB). Dans le cadre de nos expérimentations, nous avons appliqué un ratio de 90 % du temps de décision pour la résolution contre 10 % pour l'échantillonnage. Afin de justifier ce ratio, la figure 5 présente une analyse de sensibilité de MAC-UCB, en décrivant le pourcentage de victoires de MAC-UCB face à UCT pour un ratio variant de 0 % à 100 % pour la partie résolution sur l'ensemble des jeux étu-

3. Notons que ce jeu est en coopération avec les autres joueurs, donc nous présentons la moyennes obtenu pour chaque algorithme

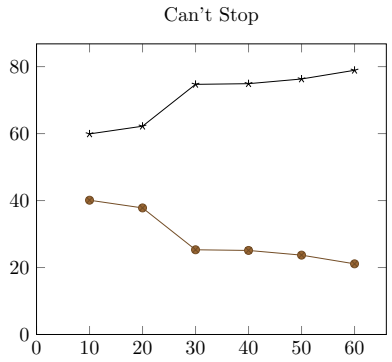


FIGURE 4 – En ordonnée, le pourcentage de victoires de MAC-UCB (×) face à UCT (○) sur *Can't Stop*, et en abscisse les différents temps possibles par coup sur 5000 instances.

diés. De manière régulière, l'optimum est atteint entre 86 et 94 %, ce qui met en lumière l'importance de résoudre les jeux en se focalisant principalement sur leur structure, imposée par les contraintes dures et le choix du seuil.

5 Conclusion

Dans cet article, nous avons exploité un fragment de la modélisation des problèmes de satisfaction par contraintes stochastiques permettant de représenter les jeux à informations complètes et incertaines. Les solutions du réseau stochastique sont interprétées comme des stratégies permettant de jouer à ces jeux. A partir de ce fragment, nous avons proposé un algorithme, MAC-UCB, qui combine résolution et simulation en utilisant à la fois une technique de programmation par contraintes pour la résolution (MAC) et une technique d'apprentissage stochastique pour la simulation (UCB). Testé dans un contexte GGP sur divers jeux avec différents temps par coup, MAC-UCB s'avère, dans la plupart des cas, être plus performant qu'UCT, la référence dans le domaine des jeux à information complètes et incertaines.

Ce travail ouvre la voie à de nombreuses perspectives de recherche. Notamment, afin d'optimiser la résolution, nous souhaitons exploiter les symétries présentes dans les μSCSP obtenus et les méthodes d'arc consistances dédiées aux SCSP (ex : [2]). De plus, il serait intéressant d'exploiter une borne de confiance plus adaptée aux problèmes de jeux (souvent

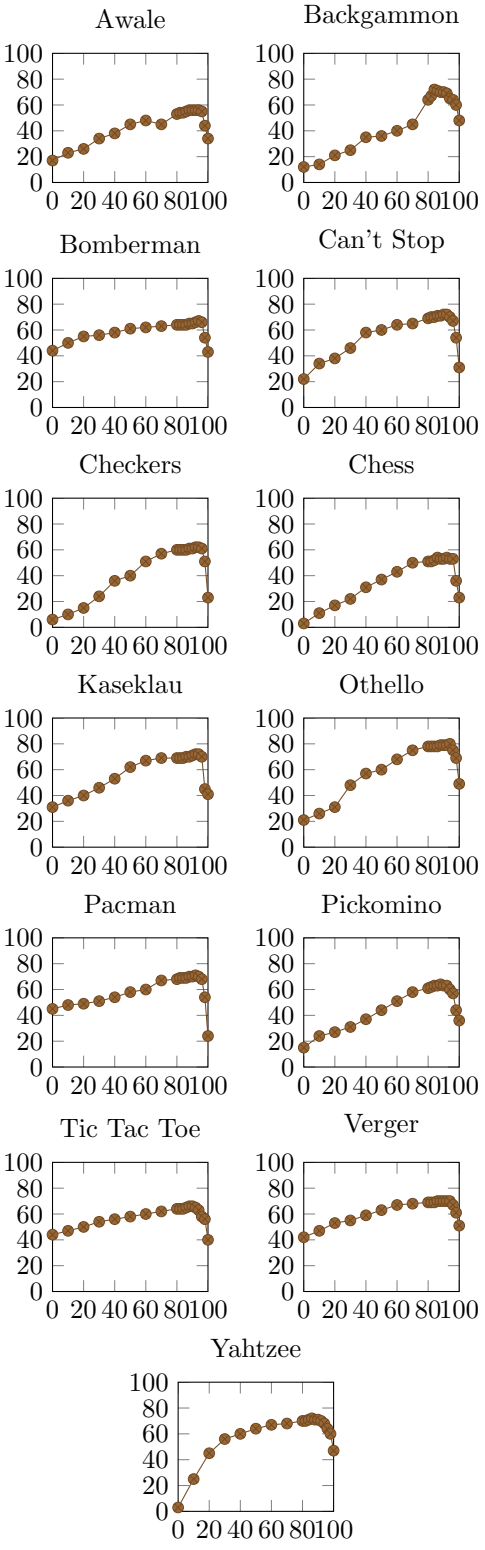


FIGURE 5 – Analyse de sensibilité entre résolution et simulation de MAC-UCB. En ordonnée, le pourcentage de victoires de MAC-UCB vs. UCT et en abscisse, le pourcentage de résolution durant les temps d'action.

compétitifs) afin d'obtenir de meilleures heuristiques. A plus long terme, nous souhaiterions étendre notre modèle afin d'intégrer les jeux à informations incomplètes.

Références

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3) :235–256, 2002.

[2] Thanasis Balafoutis and Kostas Stergiou. Algorithms for stochastic CSPs. In *Proc. of CP'06*, pages 44–58, 2006.

[3] Christian Bessiere and Guillaume Verger. Strategic constraint satisfaction problems. In *Proc. of CP'06 Workshop on Modelling and Reformulation*, pages 17–29, 2006.

[4] James Edmond Clune, III. *Heuristic evaluation functions for general game playing.* PhD thesis, University of California, Los Angeles, USA, 2008. Adviser-Korf, Richard E.

[5] R. Debruyne and C. Bessiere. Some practical filtering techniques for the constraint satisfaction problem. In *Proc. of IJCAI'97*, pages 412–417. Springer, 1997.

[6] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proc. of AAAI'08*, pages 259–264, 2008.

[7] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing : Overview of the aaai competition. *AAAI Magazine*, 26(2) :62–72, 2005.

[8] Ian P. Genta, Peter Nightingalea, Andrew Rowleya, and Kostas Stergiou. Solving quantified constraint satisfaction problems. *Artificial Intelligence*, 172(6-7) :738–77, 2008.

[9] Frédéric Koriche, Sylvain Lagrue, Éric Piette, and Sébastien Tabary. Compiling strategic games with complete information into stochastic csps. In *AAAI workshop on Planning, Search, and Optimization (PlanSOpt-15)*, 2015.

[10] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing : Game description language specification. Technical report, 2008.

[11] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8 :99–118, 1977.

[12] A.K. Mackworth. On reading sketch maps. In *Proc. of IJCAI'77*, page 598–606. Springer, 1977.

[13] Maximilian Möller, Marius Thomas Schneider, Martin Wegner, and Torsten Schaub. Centurio, a general game player : Parallel, Java- and ASP-based. *Künstliche Intelligenz*, 25(1) :17–24, 2011.

[14] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning : Theory & Practice*. Morgan Kaufmann, 2004.

[15] Thi-Van-Anh Nguyen, Arnaud Lallouet, and Lucas Bordeaux. Constraint games : Framework and local search solver. In *Proc. of ICTAI'13*, pages 8–12, 2013.

[16] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proc. of CP'94*, pages 10–20. Springer, 1994.

[17] D. Sabin and E.C. Freuder. Understanding and improving the mac algorithm. In *Proc. of CP'97*, pages 167–181. Springer, 1997.

[18] Nathan R. Sturtevant. An analysis of uct in multi-player games. *ICGA Journal*, 31(4) :195–208, 2008.

[19] Michael Thielscher. Flux : A logic programming method for reasoning agents. *Theory Pract. Log. Program.*, 5(4-5) :533–565, 2005.

[20] Michael Thielscher. A general game description language for incomplete information games. In *Proc. of AAAI'10*, pages 994–999, 2010.

[21] Toby Walsh. Stochastic constraint programming. In *Proc. of ECAI'02*, pages 111–115, 2002.