Programmation par contraintes stochastiques pour le General Game Playing avec informations incomplètes

Programmation par contraintes stochastiques pour le General Game Playing avec informations incomplètes

Frédéric Koriche Sylvain Lagrue Eric Piette *Sébastien Tabary

Université Lille-Nord de France CRIL - CNRS UMR 8188 Artois, F-62307 Lens

{koriche,lagrue,epiette,tabary}@cril.fr

Résumé

Le *game description language* avec informations incomplètes (GDL−II) est assez expressif pour représenter les jeux stochastiques multi-agents avec observation partielle. Malheureusement, une telle expressivité n'est pas possible sans un prix : le problème consistant à trouver une stratégie gagnante est $\mathrm{NExp}^{\mathrm{NP}}$-hard, une classe de complexité qui est bien au-delà de la portée des *solvers* actuels. Dans ce papier, nous identifions un fragment Pspace-complete de GDL-II, où les agents partagent les mêmes observations (partielles). Nous montrons que ce fragment peut être encapsulé dans un problème de satisfaction de contraintes stochastiques décomposable (SCSP) qui, par tour, peut être résolu en utilisant des techniques de programmation par contraintes usuelles. Dès lors, nous avons développé un algorithme de décisions séquentielles fondé sur les contraintes pour les jeux GDL−II exploitant la propagation par contraintes, l'évaluation Monte-Carlo et la détection de symétries. Notre algorithme, vérifié sur une large variété de jeux, surpasse aisément l'état de l'art des algorithmes du *general game playing* (GGP).

Abstract

The game description language with incomplete information (GDL−II) is expressive enough to capture partially observable stochastic multi-agent games. Unfortunately, such expressiveness does not come without a price : the problem of finding a winning strategy is $\mathrm{NExp}^{\mathrm{NP}}$-hard, a complexity class which is far beyond the reach of modern constraint solvers. In this paper we identify a Pspace-complete fragment of GDL−II, where agents share the same (partial) observations. We show that this fragment can be cast as a decomposable stochastic constraint satisfaction problem (SCSP) which, in turn, can be solved using general-purpose constraint programming techniques. Namely, we develop a constraint-based sequential decision algorithm for GDL−II games which exploits constraint propagation, Monte-Carlo sampling, and symmetry detection. Our algorithm, validated on a wide variety of games, significantly outperforms the state-of-the-art general game playing algorithms.

1 Introduction

De toutes les activités humaines, les jeux amènent l'un des exemples de comportement intelligent les plus illustratifs. En effet, un joueur doit faire face à de nombreuses taches complexes, telles que la compréhension de règles abstraites, l'évaluation de la situation courante, choisir le meilleur mouvement possible et finalement élaborer une stratégie gagnante. Dans le cadre de l'intelligence artificielle, le challenge *General Game Playing* (GGP) [8, 7] propose de développer des joueurs informatisés qui comprennent les règles de jeux précédemment inconnus et d'apprendre à y jouer efficacement sans intervention humaine.

Dans le *General Game Playing*, les règles d'un jeu sont décrites dans un formalisme de représentation déclaratif de haut niveau, nommé le *Game Description Language* (GDL). La première version de ce langage (GDL−I) est restreinte aux jeux déterministes avec informations complètes [14]. Tandis qu'un jeu GDL−I peut impliquer plusieurs joueurs, chaque participant a une connaissance complète de l'état courant du jeu, les actions de ces adversaires et des effets déterministes de l'ensemble des actions.

Dans le but de soulever ces restrictions, Schiffel et Thielscher [17, 18] ont récemment proposé une nouvelle version du *Game Description Language* (GDL−II) permettant de représenter les jeux avec

*Papier doctorant : Eric Piette est auteur principal.

informations incomplètes. Dans un jeu GDL-II, les joueurs peuvent avoir un accès limité aux informations de l'état courant et les effets de l'ensemble des actions sont incertains. A ce titre, GDL-II est assez expressif pour représenter les jeux stochastiques avec observation partielle (POSGs), qui couvrent une large variété de problèmes multi-agents de décisions séquentielles. Cependant, une telle expressivité est impossible sans payer le prix : le problème consistant à trouver une stratégie gagnante est NEXP^{NP}-hard [10], une classe de complexité qui est bien au-delà de la classe usuelle de complexité pour les jeux avec informations complètes (PSPACE). Bien que différents algorithmes ont été conçus pour résoudre des instances spécifiques de POSGs tel que le *Contract Bridge* [9] et *Poker* [2], ceux sont tous des programmes dédiés qui reposent fortement sur la connaissance humaine des règles du jeu et sur des fonctions d'évaluations. Par contre, le développement d'un joueur de *General Game Playing* pour POSGs semble extrêmement difficile suite à la barrière de la complexité.

Malgré ce résultat théorique, plusieurs algorithmes GGP ont été récemment développés pour résoudre certains fragments de GDL-II. Parmi eux, les jeux déterministes mono agent avec informations incomplètes [5] et les jeux stochastiques multi-agents avec informations complètes au travers de notre algorithme MAC-UCB [13]. Cette dernière approche repose sur les techniques de programmation par contraintes qui, en pratique, se sont révélées gagnantes. Notamment au cours de la dernière compétition internationale GGP dénommée *The Tiltyard Open* 2015, nous avons fait participer MAC-UCB sous le nom de *WoodStock* (We Our Own Development STOChastic toolKit) qui a terminé 2^{2ème} au cours de la phase de qualification et 3ème en phase terminale. L'ensemble des matchs sont consultables ici [1]. Notons d'ailleurs que cette compétition est dédiée aux jeux déterministes en GDL-I dans lesquels notre algorithme MAC-UCB ne peut profiter de sa principale fonctionnalité de filtrage via les variables stochastiques.

Notre présente étude a pour objectif d'étendre MAC-UCB à la gamme des jeux GDL-II. La contribution principale de cet article est résumé ainsi : (i) nous présentons un fragment important de GDL-II, où les joueurs partagent les mêmes observations (partielles) et qui peut être représenté par un SCSP ; (ii) Nous proposons un algorithme de décisions séquentielles à base de contraintes pour GDL-II qui exploitent la propagation par contraintes, l'évaluation Monte-Carlo et la détection de symétries ; (iii) Nous apportons une étude expérimentale comparative sur de nombreux jeux GDL, incluant des jeux déterministes, des jeux stochastiques et des jeux stochastiques avec observations partielles (avec informations partagées).

Les résultats expérimentaux montrent que la technique de programmation par contraintes stochastiques conduite par les symétries surpasse aisément l'état de l'art des algorithmes de *General Game Playing*.

2 GGP avec informations incomplètes

Les problèmes étudiés en GGP sont des jeux séquentiels finis. Chaque jeu implique un nombre fini de joueurs et un nombre fini d'états, incluant un état initial distinct et un ou plusieurs états terminaux. A chaque tour de jeu, chaque joueur possède un nombre fini d'actions (nommé coups légaux) ; l'état courant du jeu est mis à jour par les conséquences simultanées de l'action de chaque joueur (qui peut être *noop* pour 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. Dans un jeu stochastique, un joueur distinct, souvent faisant référence à la chance, choisit ses actions aléatoirement selon une distribution de probabilités définie sur ses coups légaux. Dans un jeu à informations incomplètes, certains aspects (nommé *fluents)* de l'état courant du jeu ne sont pas complètement révélés aux agents. Nous allons nous concentrer sur les jeux stochastiques à informations partagées dans lesquels à chaque tour l'ensemble des agents ont les mêmes (parfois incomplètes) informations sur l'état du jeu.

La syntaxe de GDL-II. GDL est un langage déclaratif permettant de représenter les jeux finis. Basiquement, un programme GDL est un ensemble de règles décrites en logique du premier ordre. Les joueurs et les objets du jeu (pièces, dés, lieux, cases, etc.) sont décrits par des constantes tandis que les fluents et les actions sont décrites par des termes du premier ordre. Les atomes d'un programme GDL sont construits selon un ensemble fini de symboles de relations et de symboles de variables. Quelques symboles ont un sens spécifique dans un programme et sont décrits dans le tableau 1. Par exemple, dans le *TicTacToe*, *legal(Alice,mark(X,Y))* indique que le joueur Alice à l'autorisation de jouer la case (X,Y) du plateau. En GDL-II, les deux derniers mots-clés du tableau sont ajoutés pour représenter les jeux stochastiques (*random*) et les jeux à informations partielles (*sees*).
Les règles d'un programme GDL sont des clauses de Horn du premier ordre. Par exemple, la règle :

$$\texttt{sees}(\texttt{Alice}, \texttt{cell}(\texttt{X}, \texttt{Y}, \texttt{o})) \leftarrow \texttt{does}(\texttt{Alice}, \texttt{mark}(\texttt{X}, \texttt{Y}))$$

1. www.ggp.org/view/tiltyard/matches/#tiltyard_open_20151204_2

Mots-clés	Description
`role(P)`	`P` est un joueur
`init(F)`	Le fluent `F` est présent à l'état initial
`true(F)`	Le fluent `F` est présent
`legal(P,A)`	Le joueur `P` peut réaliser l'action `A`
`does(P,A)`	Le joueur `P` réalise l'action `A`
`next(F)`	Le fluent `F` est présent dans le prochain état
`terminal`	L'état courant est terminal
`goal(P,V)`	Le joueur `P` obtient un score `V`
`sees(P,F)`	Le joueur `P` perçoit `F`
`random`	Le joueur "chance"

TABLE 1 – les mots-clés GDL-II

indique que Alice peut voir l'effet provoqué par le marquage des cases du plateau. Dans le but de représenter un jeu séquentiel fini, un programme GDL doit obéir à certaines conditions syntaxiques définies à l'aide de termes et de relations apparaissant dans les règles. Nous référons le lecteur à [14, 22] pour une analyse détaillée de ces conditions.

La sémantique de GDL-II. Pour un entier positif n, soit $[n] = \{1, \cdots, n\}$. Pour un ensemble fini S, soit Δ_S signifiant la probabilité sur S, c'est à dire, l'ensemble de toutes les distributions de probabilités sur S. De nombreuses descriptions de jeux à informations incomplètes sont proposées dans la litérature (voir e.g. [17]). Nous nous concentrons ici sur une variante de [5].

Formellement, un jeu stochastique avec informations partielles avec des coups légaux (POSG) est un tuple $G = \langle k, S, s_0, S_g, A, L, P, B, R\rangle$, où $k \in \mathbb{N}$ est le nombre de joueurs, S est l'ensemble fini des états, incluant l'état initial s_0, et un sous ensemble S_g d'états terminaux. A est un ensemble fini d'*actions*. Comme d'habitude le *profil d'actions* est un tuple $\boldsymbol{a} = \langle a_1, \ldots, a_k\rangle \in A^k$; par a_p, nous définissons l'action du joueur p, et par $\boldsymbol{a}_{-p}$ le profil d'actions $\langle a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_k\rangle$ des joueurs restants. $L : [k] \times S \to 2^A$ définit l'ensemble des *actions légales* $L_p(s)$ du joueur p à l'état s ; nous supposons $L_p(s) = \emptyset$ pour tous $s \in S_g$. $P : S \times A^k \hookrightarrow \Delta_S$ est la *fonction de transition partielle de probabilité*, qui attribue à chaque état $s \in S$ et chaque profil d'actions $\langle a_1, \ldots, a_k\rangle \in L(s)$ à une distribution de probabilités sur S. $B : [k] \times S \to \Delta_S$ est la *fonction de croyance* qui attribue chaque joueur $p \in [k]$ et chaque état $s \in S$ à une distribution de probabilités $B_p(s)$ sur S, représentant l'état de croyance de p à s. Finallement, $R : [k] \times S_g \to [0, 1]$ est la *fonction de score* qui attribue chaque joueur $p \in [k]$ et chaque état terminal $s \in S_g$ à une valeur $R_p(s) \in [0, 1]$, représentant le score de p sur s.

Avec ces notions en tête, le POSG G associe au programme GDL-II `G` défini comme suit. Soit `B` dénotant la base Herbrand (i.e. l'ensemble des termes) de `G` ; A (resp. F) est l'ensemble des termes d'actions (resp. fluent) apparaissant dans `B`. Nous utilisons $\mathtt{G} \models \mathtt{A}$ pour indiquer que l'atome `A` est vrai dans l'ensemble de réponses uniques de `G`. Le nombre k de termes `p` tel que $\mathtt{role(p)} \in \mathtt{G}$, indique l'ensemble des joueurs.

Chaque état s est un sous ensemble de F. Particulièrement, l'état initial s_0 est $\{f : \mathtt{G} \models \mathtt{init(f)}\}$, et tout état terminal est l'ensemble de fluents $s = \{f_1, \cdots, f_n\}$ tel que $\mathtt{G} \cup s^{\mathtt{true}} \models \mathtt{terminal}$, où $s^{\mathtt{true}}$ est l'ensemble de faits $\{\mathtt{true(f_1)}, \cdots, \mathtt{true(f_n)}\}$. L'ensemble $L_p(s)$ des actions légales pour le joueur p à l'état s est donné par $\mathtt{G} \cup s^{\mathtt{true}} \models \mathtt{legal(p,a)}$. Spécialement, $L_0(s)$ indique l'ensemble des actions légales pour le joueur "chance" (`random`). Tout profil d'actions (étendu au joueur "chance") $\boldsymbol{a} = \langle a_0, a_1, \ldots, a_k\rangle \in L_0(s) \times L_1(s) \times \cdots \times L_k(s)$ indique un successeur s' de s donné par $\{f : \mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{next(f)}\}$, où $\boldsymbol{a}^{\mathtt{does}}$ est l'ensemble de faits $\{\mathtt{does(a_0)}, \mathtt{does(a_1)}, \ldots, \mathtt{does(a_k)}\}$. La distribution de probabilités $P(s, \boldsymbol{a}_{-0})$ sur tous ces successeurs est une distribution uniforme, i.e. $P(s, \boldsymbol{a}_{-0})(s') = 1/|L_0(s)|$. L'état de croyance $B_p(s)$ du joueur p de tout successeur s' de s est donné par la distribution jointe $\prod_{f \in F} P(f)$, où $P(f) = 1$ si $\mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{sees(p,f)}$, et $P(f) = 1/2$ sinon. Finallement, le score $R_p(s)$ du joueur p à l'état terminal s est la valeur v tel que $\mathtt{G} \cup s^{\mathtt{true}} \models \mathtt{goal(p,v)}$.

Un fragment Pspace de GDL-II. Un jeu avec informations partagées représente tout jeu stochastique avec informations partielles G dans lequel à chaque tour tous les joueurs partagent le même état de croyance, i.e. $B_1(s) = \cdots = B_k(s)$ pour tout état $s \in S$. Nous utilisons ici $B(s)$ pour indiquer l'état de croyance commun dans s. Clairement, tout jeu avec informations partagées peut être converti dans un jeu stochastique complètement observable, en remplaçant la fonction de transition P et la fonction de croyance B par une nouvelle fonction de croyance $Q : S \times A^k \hookrightarrow \Delta_S$ défini ainsi :

$$Q(s, \boldsymbol{a})(s') = \prod_{t \in S} P(s, \boldsymbol{a})(t) \cdot B(t)$$

pour tout $\boldsymbol{a} \in L(s)$. En d'autres mots, $Q(s, \boldsymbol{a})(s')$ est la probabilité d'observer s' après avoir effectué le profil d'actions $\boldsymbol{a}$ à l'état s. Etant donné que pour tout jeu G stochastique avec informations partagées, une politique jointe de G est une correspondance $\boldsymbol{\pi} : S \to A^k$, où $\pi_p(s)$ est la politique du joueur p, et $\boldsymbol{\pi}_{-p}(s)$ est la politique jointe des autres joueurs. Soit un vector de seuil $\boldsymbol{\theta} \in [0, 1]^k$, nous pouvons dire que $\boldsymbol{\pi}$ est une *politique gagnante* pour le joueur p si le score espéré de p w. r. t. $\boldsymbol{\pi}$ est plus grand que θ_p.

Basé sur la notion d'informations partagées, nous pouvons maintenant examiner quelques restrictions des programmes GDL-II qui garantissent ensemble que la recherche d'une politique gagnante est dans

PSPACE. Notamment, un programme GDL-II $\mathtt{G}$ à une profondeur limitée si le nombre de termes dans l'univers d'Herbrand de $\mathtt{G}$ est polynomial sur $|\mathtt{G}|$. Si $\mathtt{G}$ est limité et que chaque règle de $\mathtt{G}$ a un nombre constant de variables, alors $\mathtt{G}$ est *propositionel*. Pour un entier T, $\mathtt{G}$ est l'*horizon* T si tout état terminal est atteignable après au plus T rounds. Pour finir, $\mathtt{G}$ est à informations partagées si pour chaque joueur p, chaque fluent f, chaque état s, et chaque profil d'actions $\boldsymbol{a}$, on a si $\mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{sees}(\mathtt{p}, \mathtt{f})$, alors $\mathtt{G} \cup s^{\mathtt{true}} \cup \boldsymbol{a}^{\mathtt{does}} \models \mathtt{sees}(\mathtt{q}, \mathtt{f})$, pour tout joueur $q \in [k]$.

Théorème 1. *Soit $\mathcal{G}_T \subseteq$ GDL-II le fragment propositionel, des programmes d'informations partagées d'horizon T. Alors, la recherche consistant à trouver une politique gagnante de $\mathcal{G}_T$ est* PSPACE*-complete.*

Preuve (Schéma) Soit $\mathcal{G}_T$ inclut les jeux stochastiques à informations complètes comme un cas particulier, le problème est PSPACE-hard. Pour tout jeu fini et à profondeur limitée $\mathtt{G} \in \mathcal{G}_T$, une politique gagnante peut être trouvée en temps $f(|\mathtt{G}|)$ et $g(|\mathtt{G}|)$ espaces utilisant les machines de Turing alternativement stochastiques, i.e. une machine de Turing qui inclut des états stochastiques (pour `random`), des états existentiels (pour le joueur p), et des états universels (pour tous les autres joueurs). Soit $\mathtt{G}$ propositionel, le nombre de fluents et le nombre d'actions sont polynomiaux sur $|G|$, qui ensemble impliquent que $g(|\mathtt{G}|)$ est polynomial. A chaque état de jeu, la machine de Turing alternativement stochastique peut deviner un profil d'actions en utilisant ces états existentiels. Soit $k \leq |\mathtt{G}|$, le prochain état de jeu pouvant être trouvé en utilisant un nombre polynomial d'états universels et d'états stochastiques. Ceci, avec le fait que la machine de Turing peut trouver un état terminal avec au plus T tours, implique que $f(|\mathtt{G}|)$ est aussi polynomial. Pour finir, soit toutes machines de Turing alternativement stochastiques utilisant un temps et un espace polynomial peut être simulé par NPSPACE (see e.g. [1]) donc, en utilisant le théorème de SavitchNPSPACE = PSPACE, il s'en suit que $\mathcal{G}_T$ est dans PSPACE, ce qui confirme le théorème. □

3 Le cadre SCSP

Empruntant la terminologie de [23], les réseaux de contraintes stochastiques étendent le standard cadre CSP en introduisant des variables stochastiques en plus des variables de décisions usuelles. Nous nous concentrons ici sur une généralisation du modèle original SCSP qui représente de multiples agents et les distributions de probabilités conditionnelles sur les variables stochastiques.

Définition 1. Un problème k-joueur de *Satisfaction de Contraintes stochastiques (SCSP)* est un 6-tuple $N = \langle V, Y, D, C, P, \theta \rangle$, tel que $V = (v_1, \cdots, v_n)$ est un tuple fini de variables, $Y \subseteq V$ est l'ensemble des *variables stochastiques*, D est une correspondance de V sur les *domaines*, C est un ensemble de *contraintes*, P est un ensemble de *tables de probabilités conditionnelles*, et $\theta \in [0,1]^k$ est un *seuil*.

- Chaque contrainte dans C est une paire $c = (scp_c, val_c)$, tel que scp_c est un sous ensemble de V, nommé la *portée* de c, et val_c est une correspondance de $D(scp_c)$ vers $([0,1] \cup \{-\infty\})^k$.
- chaque table de probabilités conditionnelles dans P est un triplet $(y, scp_y, prob_y)$, où $y \in Y$ est une variable stochastique, scp_y est un sous ensemble de variables apparaissant avant y dans V, et $prob_y$ est une correspondance de $D(scp_y)$ vers une distribution de probabilité $D(y)$.

Par X, nous indiquons l'ensemble $V \setminus Y$ de *variables de décisions*. Quand $y \in Y$ est une variable stochastique et $\tau \in D(scp_y)$ est un tuple de valeur dans la table de probabilités conditionnelles y, alors nous utilisons $P(y \mid \tau)$ qui indique la distribution $prob_y(\tau)$. En particulier, si $d \in D(y)$, alors $P(y = d \mid \tau)$ est la probabilité que y prend la valeur d donnée par τ.

Soit un sous ensemble $U = (v_1, \cdots, v_m) \subseteq V$, une *instanciation* de U est un assignement I de valeurs $d_1 \in D(v_1), \cdots, d_m \in D(v_m)$ aux variables $v_1, \cdots, v_m$, aussi écrites $I = \{(v_1, d_1), \ldots, (v_m, d_m)\}$. Une instantiation I sur U est *complète* si $U = V$. Soit un sous ensemble $U' \subseteq U$, nous utilisons $I_{|U'}$ qui indique la restriction de I vers U', c'est à dire, $I_{|U'} = \{(v_i, d_i) \in I : v_i \in U'\}$. La *probabilité* de I est donnée par :

$$P(I) = \prod_{y \in Y : scp_y \subseteq U} P(y = I_{|y} \mid I_{|scp_y})$$

Corrélativement, l'*utilité* d'une instanciation I sur U est donnée par

$$val(I) = \sum_{c \in C : scp_c \subseteq U} val(I_{|scp_c})$$

Notons que $val(I)$ est un tuple $(val_1(I), \cdots, val_k(I))$ assignant une utilité à chaque joueur. Une instanciation I est *localement cohérente* si $val_p(I) \neq -\infty$ pour chaque joueur p, c'est à dire, I satisfait chaque contrainte dure dans C. I est *globalement cohérente* (ou *cohérente*) si il peut être étendu à une instanciation complète I' qui est localement cohérente.

Une *politique* π pour le réseau N est un arbre où chaque noeud interne est étiqueté par une variable v et chaque arête est étiquetée par une valeur dans $D(v)$. Spécifiquement, les noeuds sont étiquetés selon l'ordre V : le noeud racine est étiqueté par v_1,

et chaque fils d'un noeud v_i est étiqueté par v_{i+1}. Les noeuds de décisions x_i ont un unique fils, et les noeuds stochastiques y_i ont $|D(y_i)|$ enfants. Enfin, chaque feuille dans π est étiquetée par l'utilité $val(I)$, où I est l'instanciation complète spécifiée par le chemin de la racine de π à chaque feuille. Soit $\mathcal{L}(\pi)$ l'ensemble de tous les instanciations complètes induites par π. Alors, l'*utilité attendue* de π est la somme de sa feuille d'utilité pondérée par leur probabilité. Formellement, $val(\pi) = \sum_{I \in \mathcal{L}(\pi)} P(I) val(I)$.

Une *politique* π est *faisable* si et seulement si $val(\pi) \geq (0, \ldots, 0)$, i.e. tout chemin dans π est globalement cohérent. Finalement, π est une *solution* de N si son utilité attendue est plus grande ou égale que le seuil $\theta = (\theta_1, \ldots, \theta_k)$. Cela implique que $val_p(\pi) \geq \theta_p$ pour tout joueur p.

Un *(decision) niveau* dans un SCSP est un tuple de variables $\langle X_i, Y_i \rangle$, où X_i est un sous ensemble de variables de décisions, Y_i est un sous ensemble de variables stochastiques, et de variables de décisions survenant avant toute variable stochastique [11]. Par extension :

Définition 2. Un *T-niveau k-joueur SCSP* est un k-player SCSP $N = \langle V, Y, D, C, P, \theta \rangle$, dans lequel V peut être partitionné en T *niveaux*, i.e. $V = (\langle X_1, Y_1 \rangle, \ldots, \langle X_T, Y_T \rangle)$, où $\{X_i\}_{i=1}^T$ est une partition de X, $\{Y_i\}_{i=1}^T$ est une partition de Y, et $scp_{y_i} \subseteq X_i$ pour chaque $i \in \{1, \ldots, T\}$ et chaque $y_i \in Y_i$. Si $T = 1$, N est appelé un *1-niveau* SCSP, et est défini par μSCSP.

Notons que la recherche d'une politique gagnante dans un SCSP est PSPACE-complète. Le problème reste dans PSPACE pour T-niveau k-joueurs SCSPs, car chaque nievau du problème est dans $\mathrm{NP}^{\mathrm{PP}}$.

SCSP pour GDL-II. En se basant sur ce cadre, la traduction d'un programme k-joueur $\mathtt{G} \in \mathcal{G}_T$ vers un T-niveau k-joueur SCSP est spécifié comme suit. Nous commençons par convertir G vers un ensemble de règles closes G′, dont leur taille est polynomial dans |G| car G est propositionel. Pour G′ nous associons un 1-niveau SCSP $N = \langle g_t, \{f_t\}, \{a_t\}, y_t, \{f_{t+1}\} \rangle$, où g_t est une variable bouléenne indiquant si le jeu a atteint un état terminal ; $\{f_t\}$ et $\{f_{t+1}\}$ sont les ensembles de variables stochastiques décrivant l'état du jeu au tour t et $t+1$, respectivement ; $y_t = a_{t,0}$ est une variable stochastique décrivant l'ensemble des coups légaux du joueur "chance" ; et $\{a_t\} = \{a_{t,1}, \ldots, a_{t,k}\}$ est l'ensemble des variables de décisions, chaque $a_{t,p}$ décrit l'ensemble des coups légaux du joueur p. A proprement parler, N n'est pas un 1-niveau SCSP, en raison des variables stochastiques $\{f_t\}$, mais le domaine de ces variables est un singleton, indiquant l'état courant du jeu. Ainsi, lors du processus de résolution, $\{f_t\}$ peut être supprimé par N pour produire un μSCSP.

Les clauses de Horn d'un programme GDL G peuvent être naturellement partitionnées en règles `init` décrivant l'état initial, en règles `legal` spécifiant les coups légaux de l'état courant, en règles `next` indiquant les effets des actions, en règles `sees` décrivant les observations de chaque joueur sur le jeu, et en règles `goal` définissant les scores des joueurs à l'état terminal. Les règles `init`, `legal` et `next` sont encodées en contraintes dures dans le réseau N. Les règles `sees` sont utilisées pour exprimer la table de probabilités conditionnelles de chaque variable stochastique f_{t+1}. La dernière variable stochastique y_t est associée à la distribution de probabilité uniforme sur les coups légaux du joueur "chance". La contrainte relation est extraite de la même manière que les domaines des variables, par identification de toutes les combinaisons autorisées des constantes. Similairement, les règles `goal` sont encodées par une contrainte souple sur N ; basé sur leurs sémantiques, les scores des joueurs sont toujours à 0 pour les états non terminaux et une valeur de $[0, 1]$ pour tout état terminal.

En répétant T fois ce processus de conversion, nous obtenons alors un T-niveau SCSP encodant le jeu G à T-horizon. Le seuil θ peut être ajusté selon la stratégie désidée : commençant par la valeur $\boldsymbol{\theta} = (0, \ldots, 0)$ qui autorise toutes les politiques faisables, on peut utiliser la valeur attendue comme une solution du SCSP courant comme un nouveau seuil, qui détermine un SCSP plus contraint défini sur les mêmes contraintes.

4 MAC-UCB-SYM

Sur la base du fragment de SCSP pour les jeux GDL, nous présentons maintenant notre technique de résolution dénommée MAC-UCB-SYM, une extension de l'algorithme MAC-UCB [13] qui exploite la détection de symétrie. Comme indiqué ci-dessus, le réseau de contraintes stochastiques d'un programme GDL est une séquence de μSCSPs, chacun associant au tour du jeu t dans $\{1, \ldots, T\}$. Pour chaque $\mu\mathrm{SCSP}_t$ dans $\{1, \ldots, T\}$, MAC-UCB recherche l'ensemble des politiques faisables en divisant le problème en 2 parties : un CSP et un μSCSP (plus petit que l'original). La première partie est résolue avec l'algorithme MAC et la seconde partie via l'algorithme FC dédié au SCSP. Puis, un échantillonnage avec borne de confiance est réalisé pour estimer l'utilité attendue pour chaque solution réalisable d'un $\mu\mathrm{SCSP}_t$. Les symétries sont exploitées pour éviter l'exploiration inutile de μSCSPs : à chaque μSCSP résolu, les μSCSPs symétriques sont calculés et stockés avec leurs utilités attendues.

Technique de filtrage utilisant les symétries. Rappelons que la décision séquentielle associée à un jeu de stratégie est un problème d'optimisation. Classiquement, ce problème est abordé en recherchant une solution à une séquence de problèmes de satisfaction stochastiques. En pratique, à chaque tour notre joueur doit jouer, un laps de temps est dédié au choix de la prochaine action. Pendant ce temps, nous résolvons autant de μSCSPs que possible, c'est pourquoi éviter de résoudre des μSCSPs inutilement est un point clé. Comme les μSCSPs correspondent à une représentation des états du jeu avec les actions possibles de chaque joueur, il est possible de rencontrer des μSCSPs similaires qui correspondent à des états du jeu similaires. Par exemple, dans un *TicTactoe*, une fois que l'adversaire à marqué le centre, marqué l'un des quatres coins revient à une situation similaire. Dans notre approche, nous proposons de détecter et de stocker les μSCSPs symétriques dans le but de bénéficier de ceux déjà résolus.

Rappelons qu'une symétrie dans un ensemble D est une permutation sur D ou, de manière équivalente une bijection σ de D vers D. Couramment, de telles permutations peuvent être représentées comme un ensemble de *cycles*. Par exemple, si $D = \{1, 2, 3, 4\}$, alors la permutation σ spécifie que $\sigma(1) = 2, \sigma(2) = 3, \sigma(3) = 1$ et $\sigma(4) = 4$, peuvent aussi être décrites par $\{(1, 2, 3), (4)\}$, ou de manière encore plus compacte $\{(1, 2, 3)\}$ où les cycles unaires sont implicites. De nombreuses types de symétries ont été proposées dans la litérature de la programmation par contraintes, notamment les *symétries de solutions* qui préservent l'ensemble des solutions(e.g. [12, 16]), et *constraint symmetries* qui préservent l'ensemble des contraintes [4].

MAC-UCB-SYM exploite les symétries de contraintes, charactérisées par les automorphismes de la microstructure complémentaire du réseau. Plus précisément, chaque μSCSP P au niveau t est spécifié selon la Section 3, est enrichi par une contrainte dure de portée $(\{f_t\}, \{a_t\}, y_t)$, où la relation associée est formée de tuples de la forme $(s, \boldsymbol{a})$, s définis sur $\{f_t\}$ et $\boldsymbol{a}$ le profil d'actions défini sur $\{a_t\}$ et y_t. Chaque tuple $(s, \boldsymbol{a})$ est vu comme un "nogood" si tout politique depuis s avec le profil d'actions $\boldsymbol{a}$ est prédite par UCB comme une "non-solution" (son utilité espérée est sous le seuil $\boldsymbol{\theta}$). Basé sur ce μSCSP P, nous calculons les automorphismes de la microstructure dérivés de P. La microstructure est un hypergraphe, qui peut aussi être décrit par un graphe biparti : la partie gauche étant l'ensemble des litéraux (paires de variable-valeur), la partie droite est l'ensemble des tuples incohérentes de toutes les contraintes de μSCSP, et une arête correspond à l'occurence d'un litéral dans un tuple d'une contrainte. A partir de ce graphe, nous commençons par calculer les automorphimes et nous générons les μSCSPs symétriques. Puis, ces μSCSPs sont stockés avec leurs utilités attendues dans une base (en pratique une table de *hash*). Avant de résoudre un μSCSP donné, nous recherchons dans la base si le μSCSP correspond à un μSCSP symétrique déjà rencontré. Si c'est le cas, l'utilité associée est directement obtenu sans le résoudre.

Mise en pratique. Considérons un *TicTacToe* 2×2 contre un joueur jouant aléatoirement. Un joueur gagne si il remplit une ligne ou une colonne (les diagonales sont exclues). MAC-UCB-SYM joue avec le symbole "o". Supposons que le symbole "x" est déjà présent en $(1, 1)$ à l'état initial. La Figure 1 représente l'arbre de recherche obtenu par MAC-UCB-SYM, dans lequel une symétrie est apparue entre l'action `mark(1,2)` (en bleu) et l'action `mark(2,1)` (en rouge). Après avoir résolu l'état initial μSCSP, MAC-UCB-SYM résout μSCSP_1^1 obtenu par l'action `mark(1,2)`. UCB renvoit une utilité attendue de 0.25 pour cet état. Après avoir calculé l'automorphime pour μSCSP_1^1, nous générons et stockons le μSCSP symétrique correspondant à la permutation $\{(cell, 1\_2\_o), (cell, 1\_2\_blank)\}$, $\{(cell, 2\_1\_blank), (cell, 2\_1\_o)\}$,$\{(action_o, 1\_2), (action_o, 2\_1)\}$. Puis MAC-UCB-SYM s'intéresse à μSCSP_1^2 : ici, la détection de symétrie ne révèle pas de nouveau automorphisme et UCB associe une utilité attendue de 0 pour ce μSCSP. Enfin le dernier μSCSP μSCSP_1^3 correspond à un μSCSP symétrique généré précédemment. Par conséquent, la même utilité 0.25 associée à μSCSP_1^1 est associée à μSCSP_1^3 sans lancer UCB.

MAC-UCB-SYM choisit une action parmi ceux avec l'utilité attendue la plus forte : ici, il sélectionne aléatoirement l'une des actions `mark(1,2)` ou `mark(2,1)`. Ainsi, l'idée est de jouer sur la même ligne ou la même colonne que le symbole "x" pour espérer un nul.

5 Résultats Expérimentaux

Dans ce cadre, nous présentons une série de résultats expérimentaux réalisés sur un *cluster* d'Intel Xeon E5-2643 CPU 3.3 GHz avec 64 GB de mémaoire RAM et quatres coeurs sour Linux. Notre algorithme a été implémenté en C++ et nous utilisons NAUTY [15] pour la détection de symétrie.

Nous avons sélectionné 20 jeux déterministes décrits en GDL extraits de la base de jeux GDL du serveur Tiltyard [2], et 15 jeux stochastiques en GDL-II incluant 5 sans règles `sees`. La majorité des jeux GDL-II a été sélectionné du serveur Dresden [3]. Les expérimentations

2. http://tiltyard.ggp.org/
3. http://ggpserver.general-game-playing.de/

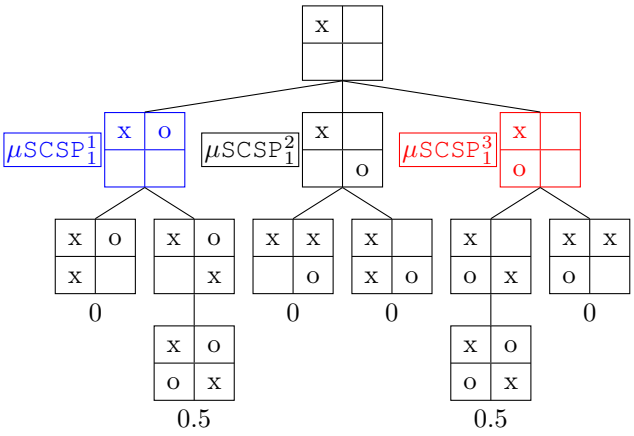

FIGURE 1 – L'arbre de recherche du *TicTacToe* 2 × 2 avec les scores obtenus par MAC-UCB-SYM pour chaque état terminal

ont été réalisées sur une grande variété de jeux pour un total de 40, 000 matchs. Il est possible de retrouver plus en détail chaque jeu sur le site boardgamegeek.com.

Contexte. Les différentes compétitions de jeux sont organisées entre les différents joueurs de *General Game Playing*. Le premier joueur est une version multi-joueurs de l'algorithme UCT [21], qui représente l'état de l'art des joueurs GGP pour les jeux déterministes. Le second joueur est Sancho version 1.61c [4], Un joueur basé sur la recherche Monte-Carlo dans les arbres de recherche élaboré par S. Draper and A. Rose, qui a gagné l'*International General Game Playing Competition* 2014 (IGGPC'14). Le troisième joueur est l'algorithme GRAVE de [3], qui implémente la technique *Generalized Rapid Action Value Estimation technique*, une généralisation de la méthode RAVE [6] adaptée pour GGP. Enfin, nous comparons aussi notre joueur à CFR [20], le *CounterFactual Regret technique*, qui recherche des équilibres de Nash et les suit afin de jouer. CFR est considéré ici comme l'état de l'art pour les jeux GDL-II avec informations incomplètes car l'algorithme HyperPlayer-II [19] n'était pas présentement utilisable au cours des expérimentations.

Pour l'ensemble des matchs, nous utilisons les paramètres de la compétition Tiltyard Open 2015 (la dernière compétition internationale de GGP) : 180 secondes avant le début du jeu et 15 à chaque tour pour jouer.

Afin de comparer notre algorithme MAC-UCB-SYM avec sa dernière version MAC-UCB, nous réalisons une analyse de sensibilité pour résoudre le dilemme entre exploitation (la partie résolution) et exploration (la partie évaluation) mais aussi avec le temps dédié à la détection de symétrie pour MAC-UCB-SYM.

Dans nos expérimentations, 75 % du temps de délibération est dédié à l'exploitation et 25 % à l'exploration pour MAC-UCB. Pour MAC-UCB-SYM, 50 % du temps délibération est dédié à l'exploitation, 25 % à l'exploration et 25 % à la détection de symétrie. Afin de justifier ces ratios, la Figure 2 rapporte l'analyse de sensibilité des 2 algorithmes en moyenne sur l'ensemble des jeux sélectionnés. Le premier graphique indique le score obtenu en moyenne par MAC-UCB contre chaque joueur pour un ratio évoluant entre 0 % et 100 % pour la partie de résolution. L'optimale est atteint entre 72 et 78 %. Le second graphique présente la même valeur pour MAC-UCB-SYM. La partie concernant l'exploitation (UCB) est fixée à 25 % et le temps dédié à la détection de symétrie évolue entre 0 % et 75 %. Le meilleur score obtenu en moyenne est atteint entre 23 % et 27 %.

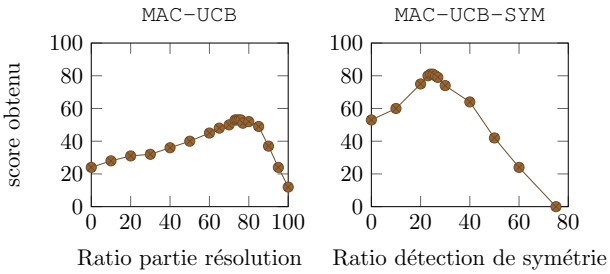

FIGURE 2 – Analyse de sensibilité pour MAC-UCB et MAC-UCB-SYM permettant de paramétrer l'exploitation, l'exploiration et la détection de symétrie.

Pour l'ensemble des jeux GDL-I, nous réalisons 100 matchs entre MAC-UCB-SYM et chacun des autres joueurs. Le nombre de matchs pour les jeux GDL-II a été fixé à 400. Pour un soucis d'équitabilité, les roles des joueurs sont échangés à chaque match.

Nos résultats sont résumés dans le Tableau 2. Les lignes sont regroupées en 4 parties, respectivement répartissant les jeux GDL-I, les jeux GDL-II sans règles *sees*, les jeux GDL-II en solo avec règles *sees* et les jeux GDL-II à informations partagées (avec les mêmes règles *sees* pour tous les joueurs). La colonne moy :Nogoods indique le nombre moyen de *nogoods* trouvé au cours de l'analyse de la microstructure complément de chaque μSCSP. La colonne #Sym représente le nombre moyen d'automorphismes détectés par MAC-UCB-SYM. La colonne Ame est le ratio entre les tailles des arbres de recherche entre MAC-UCB-SYM et MAC-UCB. Finalement, les dernières colonnes indiquent le pourcentage moyen de victoires de MAC-UCB-SYM contre l'adversaire sélectioné. Par

4. http://sanchoggp.github.io/sancho-ggp/

GDL-I								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB**	**UCT**	**CFR**	**Sancho**	**GRAVE**
Amazons torus 10x10	193,759.44	256	84.50 %	81	96	95	81	76
Battlebrushes	1309,262.32	850	77.01 %	46	76	56	47	53
Breakthrough suicide	24,591.15	124	84.54 %	92	93	95	78	58
Chess	304,993.64	328	83.97 %	77	94	90	87	86
Connect Four 20x20	685,938.7	616	91.02 %	86	98	96	96	62
Connect Four Simultaneous	132,192.04	104	65.16 %	74	89	64	81	74
Copolymer with pie	192,525.23	34	70.24 %	74	87	88	76	80
Dots and boxes suicide	115,705.92	224	59.68 %	63	96	95	86	80
English Draughts	310,185.31	216	82.60 %	84	98	98	58	66
Free For All 2P	51,889.93	58	19.35 %	52	84	82	71	59
Hex	655,047.62	840	71.18 %	84	100	100	79	72
Knight Through	718,259.40	112	71.35 %	81	97	95	87	80
Majorities	473,112.80	670	75.85 %	83	97	94	84	80
Pentago	467,806.70	69	15.97 %	52	84	73	52	70
Quarto	548,426.11	52	12.28 %	53	86	71	56	64
Shmup	137547.01	32	33.33 %	56	86	76	51	53
Skirmish zero-sum	212,782.04	378	83.02 %	84	98	99	67	79
TicTac Chess 2P	661,546.67	143	76.64 %	95	98	94	85	74
TTCC4 2P	701,739.72	84	86.61 %	82	97	98	68	60
Reversi Suicide	334,927.41	256	61.36 %	71	100	100	57	62
GDL-II sans règles *sees*								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB**	**UCT**	**CFR**	**Sancho**	**GRAVE**
Backgammon	272,034.67	318	88.40 %	91.2	98.7	94.1	100	68
Can't Stop	630,456.94	847	89.06 %	88.5	93.5	88.5	100	92
Kaseklau	86,342.57	384	58.68 %	72.1	69.4	78.4	86.4	67
Pickomino	142,420.03	248	84.50 %	74.6	73.7	71.1	91.4	86.4
Yahtzee	256,497.69	340	91.48 %	86.7	86.4	89.4	90.4	62.4
GDL-II en solo avec des règles *sees*								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB-SYM**	**MAC-UCB**	**UCT**	**CFR**	**GRAVE**
GuessDice	0	0	0.00 %	15.6	15	15.7	16	16.5
MasterMind	4	4	0.00 %	63.9	67.8	53.8	68.1	60.1
Monty Hall	0	0	0.00 %	65.1	65.2	62.5	63.1	64.3
Vaccum Cleaner Random	164,875.58	324	83.79 %	74.8	61.5	34	46	58.8
Wumpus	369,117.69	128	78.95 %	71.2	32.1	40.1	44.1	51.2
GDL-II avec la même règle *sees* pour tous								
Jeu	**moy :Nogoods**	**#Sym**	**Ame**	**MAC-UCB**	**UCT**	**CFR**	**GRAVE**	
Pacman	81,704.31	244	20.63 %	62.3	61.2	64.2	60.4	
Schnappt Hubi	137,845.47	314	94.10 %	79.3	84.4	76.2	74.7	
Sheep & Wolf	54,729.44	212	61.00 %	84.1	90.2	74.3	73.2	
TicTacToe Latent 3P 10x10	919,414.21	678	79.23 %	84.5	95.1	91.7	84.6	
War (card game)	32,294.47	51	56,80 %	63.2	71.3	68.4	62.4	

TABLE 2 – Les résultats de MAC-UCB-SYM.

exemple, la case de la colonne UCT pour les échecs (la 4ème ligne) montre, que MAC-UCB-SYM gagne en moyenne dans 94% des matchs contre UCT. Sachant que le 3ème groupe ne comportent que des jeux à un joueur, les colonnes des joueurs (incluant MAC-UCB-SYM) rapportent le nombre moyen de fois où le jeu est résolu divisé par le nombre total de matchs.

Les résultats en GDL-I. Concernant les jeux déterministes, MAC-UCB-SYM surpasse clairement sa version sans symétrie. Ceci peut s'expliquer par la significative amélioration obtenue dans le nombre de noeuds explorés par MAC-UCB-SYM : une amélioration de plus de 80 % est observé dans les plus gros jeux, indiquant que la détection de symétrie est inestimable pour de tels jeux. Les quelques exceptions peuvent s'expliquer d'elles-même. Le *Free For All 2P*, le *Pentago* et le *Quarto* sont des jeux dont l'arbre de recherche n'est pas très important et où les symétries ne sont pas fréquentes ce qui explique la faible amélioration d'exploration de l'arbre de recherche sur MAC-UCB. Concernant les jeux à 4 joueurs : Le *Battlebrushes* dont l'objectif est de peindre le plus de cases possibles en 20 tours avant les 3 autres adversaires, ici la meilleure stratégie est atteinte très rapidement car l'arbre qui est de petite taille tout comme le *Shmup*, un *shoot'em up* possédant un horizon faible de seulement 20 tours. Pour toutes ces exceptions MAC-UCB-SYM obtient un score moyen semblable à son prédécesseur car l'arbre de recherche est rapidement complètement exploré par les algorithmes GGP. UCT et CFR sont vaincus par MAC-UCB-SYM, avec parfois un score obtenu de 100 % (voir le jeu *Hex*). Sancho, le *leader* de IGGPC'14, est en moyenne moins compétitif que MAC-UCB-SYM spécifiquement pour les jeux impliquant un arbre de recherche important. Nous pouvons également noter certains exemples notables comme l'*Amazons torus 10x10*, les échecs et le Puissance Quatre 20x20. Enfin GRAVE est le meilleur rival de MAC-UCB-SYM pour les jeux déterministes, mais notre algorithme obtient un score moyen meilleur pour cette catégorie.

Les résultats en GDL-II. Pour les jeux GDL-II sans règles *sees*, les quatres adversaires sont battus avec un score supérieur à 70 % pour MAC-UCB-SYM contre Sancho, UCT et CFR et supérieur à 60 % contre GRAVE. Il est possible de comprendre ce phénomène car, dans les jeux stochastiques, il est typique de rencontrer de nombreuses symétries sur les états probabilistes générées par la participation d'un joueur "chance" (simulant les dés ou les cartes). Un exemple remarquable est le jeu du *Yahtzee*, où une amélioration de plus de 90 % est observée (contre MAC-UCB), à l'aide de la détection de symétrie.

Pour les jeux avec informations partielles, Sancho ne peut pas participer car il est dédié à GDL-I (modulo une possible simulation du joueur "chance"). Pour les jeux à un joueur avec informations incomplètes, seuls le *Vaccum Cleaner* et le *Wumpus* sont assez grands pour exploiter les symétries. MAC-UCB-SYM est le meilleur joueur pour ces jeux, et réalise une réelle amélioration sur MAC-UCB. Cependant, il est parfois surpassé par CFR sur le *Mastermind* et le *MontyHall*. Notons qu'aucune symétrie n'est détectée dans le jeu *GuessDice* (où il faut en un tour deviner la valeur d'un dé) et au *MontyHall*. De plus les résultats pour le *Mastermind* sont assez peu significatifs suite à la petite taille de l'arbre de recherche.

Enfin, les 5 derniers jeux sont des jeux à informations partielles mais partagées entre les joueurs. Par exemple, le jeu *TicTacToe Latent 3P 10x10* implique 3 joueurs (en incluant le joueur "chance"). Le joueur "chance" place aléatoirement une croix ou un rond sur l'une des cases libres et les 2 autres joueurs ne peuvent le remarquer uniquement en essayant de placer par chance, leur symbole respectif sur ces cases. Le *Schnappt Hubi* (l'attrape fantôme) et le *Sheep & Wolf* sont des jeux coopératifs, où l'ensemble des agents partagent leurs observations dans le but de vaincre le joueur "chance". En dehors du *Pacman*, il est possible d'observer que la détection de symétrie pour les POSGs est aussi payante, caractérisée par une réduction substantielle de l'arbre de recherche. MAC-UCB-SYM gagne avec un score moyen supérieur à 60 % contre chaque adversaire, et particulièrement plus, avec un score moyen de 84 % pour le *TicTacToe Latent Random 10x10*.

6 Conclusion

Dans ce papier, nous avons identifié un fragment important des jeux à informations incomplètes qui peut être représenté en SCSPs, et qui peut être résolu avec des techniques usuelles de la programmation par contraintes. Notre algorithme de décisions séquentielles MAC-UCB-SYM pour les jeux GDL-II exploite la propagation par contraintes, l'évaluation Monte-Carlo et la détection de symétrie. Basé sur de nombreuses expérimentations impliquant de nombreux types de jeux et de joueurs GGP, nous avons montré que les techniques usuelles de programmation par contraintes portent leurs fruits. Notamment, la détection de symétrie offre une amélioration important pour la résolution des jeux à informations incomplètes.

A la lumière de ces résultats expérimentaux, un axe de recherche est l'étude théorique de la détection de symétrie pour des algorithmes basés sur UCB tel que UCT et GRAVE.

Références

[1] Edouard Bonnet and Abdallah Saffidine. On the complexity of general game playing. In *Proceedings of CGW*, pages 90–104, 2014.

[2] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218) :145–149, 2015.

[3] Tristan Cazenave. Generalized rapid action value estimation. In *Proceedings of IJCAI 2015*, pages 754–760, 2015.

[4] Davis Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3) :115–137, 2006.

[5] Florian Geißer, Thomas Keller, and Robert Mattmüller. Past, present, and future : An optimal online algorithm for single-player GDL-II games. In *Proceedings of ECAI*, pages 357–362, 2014.

[6] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of ICML*, pages 273–280, 2007.

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

[8] Michael R. Genesereth and Michael Thielscher. *General Game Playing*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2014.

[9] Matthew L. Ginsberg. GIB : imperfect information in a computationally challenging game. *J. Artif. Intell. Res. (JAIR)*, 14 :303–358, 2001.

[10] Judy Goldsmith and Martin Mundhenk. Competition adds complexity. In *Proceedings of NIPS*, pages 561–568, 2007.

[11] Brahim Hnich, Roberto Rossi, S. Armagan Tarim, and Steven Prestwich. Filtering algorithms for global chance constraints. *Artificial Intelligence*, 189 :69–94, 2012.

[12] Tom Kelsey, Steve Linton, and Colva M. Roney-Dougal. New developments in symmetry breaking in search using computational group theory. In *Proceedings of AISC'04*, pages 199–210, 2004.

[13] Frédéric Koriche, Sylvain Lagrue, Éric Piette, and Sébastien Tabary. General game playing with stochastic CSP. *Constraints*, 21(1) :95–114, 2016.

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

[15] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0) :94 – 112, 2014.

[16] Jean François Puget. Automatic detection of variable and value symmetries. In *Proceedings of CP'05*, pages 475–489, 2005.

[17] Stephan Schiffel and Michael Thielscher. Reasoning about general games described in gdl-ii. In *Proceedings of AAAI 2011*, pages 846–851. AAAI Press, 2011.

[18] Stephan Schiffel and Michael Thielscher. Representing and reasoning about the rules of general games with imperfect information. *J. Artif. Intell. Res. (JAIR)*, 49 :171–206, 2014.

[19] Michael John Schofield and Michael Thielscher. Lifting model sampling for general game playing to incomplete-information models. In *Proceedings of AAAI 2015*, pages 3585–3591, 2015.

[20] Mohammad Shafiei, Nathan Sturtevant, and Jonathan Schaeffer. Comparing uct versus cfr in simultaneous games. In *IJCAI Workshop on General Game Playing*, 2009.

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

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

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