Stage Master Recherche

Bibliographie

Reconnaissance automatique de l'intention d'autres agents en situation d'adversité

Domaine: Intelligence Artificielle - Théorie des jeux

Auteur:
Nathan Canva

Superviseurs:
Froduald Kabanza
Mariane Maynard
Laboratoire Planiart

ENSSAT
LANNION

Abstract: La reconnaissance de plan consiste à observer un protagoniste évoluer dans un environnement et se baser sur les observations faites pour prédire ses actions à venir, et identifier quels sont ses buts. Dans cette bibliographie, nous discuterons des différentes méthodes ayant fait l'objet de recherches dans ce domaine. Grammaire hors contexte, modèle markovien, et réseaux de neurones permettent d'apprécier le problème sous différents angles au travers d'études et d'expérimentations que l'on peux appliquer à des domaines tels que le jeu vidéo.
Le but est de pouvoir tester et utiliser différentes approches lors de mon stage, afin de pouvoir en partant de systèmes de reconnaissances de plan déjà établis, les contextualiser dans des situations d'adversité à 2 joueurs.

Table des matières

1 Introduction

La reconnaissance de plan est un domaine de recherche qui prend de l'ampleur grâce à la digitalisation de notre monde. Pour assister les utilisateurs humains dans leurs différentes tâches, détecter des cyberattaques, ou déjouer les plans d'un adversaire, il est nécessaire de comprendre quels sont leurs plans et leurs buts. Cette problématique est définie ainsi : « La reconnaissance de plan est le problème d'inférer des buts et des plans d'un agent à partir d'observations partielles de son comportement » [Cohen et al., 1981] [Pentney et al., 2006]. Ce problème change drastiquement en fonctions des postulats que l'on prend, et s'adapte donc à certains contextes et plus difficilement à d'autres.

On distingue généralement trois types de reconnaissance de plan. En premier lieu, il y a la reconnaisance de plan dite *'intended'* (intentionnée), où à la manière d'une explication ou d'un tutoriel le protagoniste observé prend en compte l'observateur et tâche de lui faire comprendre ses intentions. En deuxième lieu, il y a la reconnaissance de plan dite *'keyhole'*[Pentney et al., 2006], où l'agent observé ne sait pas qu'il est observé et exécute ses plans de manière indépendante. Enfin, il y a celle dite *'adversarial'*, où l'observé tente de tromper l'observateur sur ses réelles intentions tout en poursuivant un but. Ce type de reconnaissance de plan sert notamment dans les jeux vidéo.

Dans le jeu vidéo, il est important de comprendre les plans des autres joueurs afin de maximiser ses gains. Il faut donc prévoir correctement leurs actions futures en fonction des observations que nous avons, pour optimiser les nôtres [Aumann, 1989]. D'autres études ont été réalisées dans différents types de jeux vidéo : action et aventures [Gold, 2010], mondes ouverts [Ha et al., 2011], ou éducatifs [Min et al., 2014]. Pour chaque problème, il existe un ensemble de contraintes différentes, et donc leurs approches de résolution diffèrent également.

Les problèmes de reconnaissance de plan se distinguent également par le nombre de plans que l'agent peut poursuivre, s'il a la possibilités d'en abandonner, si certaines actions peuvent mener à la réalisation de plusieurs plans en même temps, si ces actions ont des contraintes de temps et/ou d'ordre, ou encore si l'environnement observable est partiel ou complet. Différentes approches proposent donc des solutions plus ou moins souples et adaptables en fonction du domaine d'application.

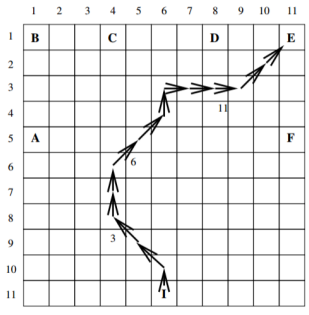


FIGURE 1 – Exemple où l'on cherche à déterminer la destination d'un agent partant de I et pouvant atteindre A, B, C, D, E, et F. Illustration utilisée dans [Ramırez and Geffner, 2010].

Dans cette bibliographie, nous étudierons d'abord le système PHATT [Geib and Goldman, 2009], qui se base sur des arbres de dérivations de grammaires hors contexte en prenant des hypothèses peu contraignantes. C'est là-dessus que repose le système HICOR [Kabanza et al., 2010] que nous allons ensuite présenter, qui introduit la reconnaissance de plans dans les jeux RTS (Real Time

Strategy). Nous verrons aussi comment l'on peut utiliser les réseaux de neurones dans le cadre de la reconnaissance de plans pour modéliser des plans [Bisson et al., 2015], identifier des habitudes de jeu [Min et al., 2014], ou encore choisir un coup à jouer. Subséquemment nous étudierons la planification inverse qui consiste à se mettre à la place de l'agent qu'on observe afin de prédire ses buts. Enfin, nous verrons la structure d'AlphaGo [Silver et al., 2016], la première IA (Intelligence Artificielle) capable de surpasser les humains aux jeux de Go.

2 État de l'art

2.1 L'approche souple des grammaires hors-contexte

Geib et Goldman ont présenté en 2009 le système PHATT (Probabilistic Hostile Agent Task Tracker) qu'ils ont construit [Geib and Goldman, 2009]. PHATT est un agent de reconnaissance de plan basé sur un modèle bayésien exploitant des arbres de dérivation de grammaires hors contexte légèrement modifiées.
Le problème des systèmes de reconnaissance de plans précédents à PHATT est qu'ils se basaient sur des hypothèses qui sont trop contraignantes dans la plupart des applications concrètes. Ces hypothèses sont qu'un agent poursuit un seul plan à la fois, que les plans se déroulent selon un ordre total, qu'il n'y a pas de relation temporelle entre les actions, que ces dernières ne sont pas paramétrées, et enfin qu'échouer d'observer une action suppose soit une observabilité partielle et arbitraire des actions de l'observé, soit que l'action n'a jamais eu lieu. PHATT ne repose sur aucune de ces hypothèses, et prend comme postulat qu'un protagoniste cherchant à atteindre un but exécutera les actions débloquées par ses actions précédentes et que ses actions sont cohérentes avec la réalisation de son objectif. En revanche, il fait l'hypothèse que l'agent ne cherche pas à tromper l'observateur sur ses réelles intentions.

PHATT travaille sur un ensemble d'actions effectuées par un agent que nous avons observé, et un ensemble contenant les actions qu'il peut réaliser qui sont consistantes avec la réalisation des plans (*pending set*). Il se base sur un mode d'exécution de plan, c'est-à-dire qu'il tente de trouver des explications à chaque observation et en déduit les plans possibles en cours d'exécution. En fonction des plans auxquels les actions appartiennent, il en déduit les probabilités d'exécution des différents plans.
En partant de la probabilité à priori que tel plan soit exécuté P(plan), et ensuite qu'une action soit observée en fonction du plan qui est exécuté P(obs|plan), nous pouvons obtenir, grâce à la règle de Bayes, ce que nous cherchons : P(plan|obs). PHATT va tenter d'expliquer pourquoi une telle action a été observée et les explications sont valables pour plusieurs plans. Ainsi, la probabilité d'un but sachant une séquence d'observations est la somme des probabilités des explications sachant ces observations, qui satisfont ce but.

$$P(goal|obs) = \sum_{\{exp_i|goals \in exp_i\}} P(exp_i|obs)$$

Les plans sont formalisés sous forme d'arbre de grammaire hors contexte. Les non-terminaux de la grammaire sont divisés en deux ensembles TNT et NNT : l'un s'occupe des terminaux liés

directement aux non terminaux et le second s'occupe des buts et du reste. Tous les terminaux et non terminaux à droite d'une règle de dérivation ne sont pas ordonnés. Les deux ensembles permettent de représenter les deux types de noeuds de nos arbres : les noeuds ET et les noeuds OU. Avec notre grammaire, nous établissons des règles de dérivations et pouvons donc formaliser nos plans.

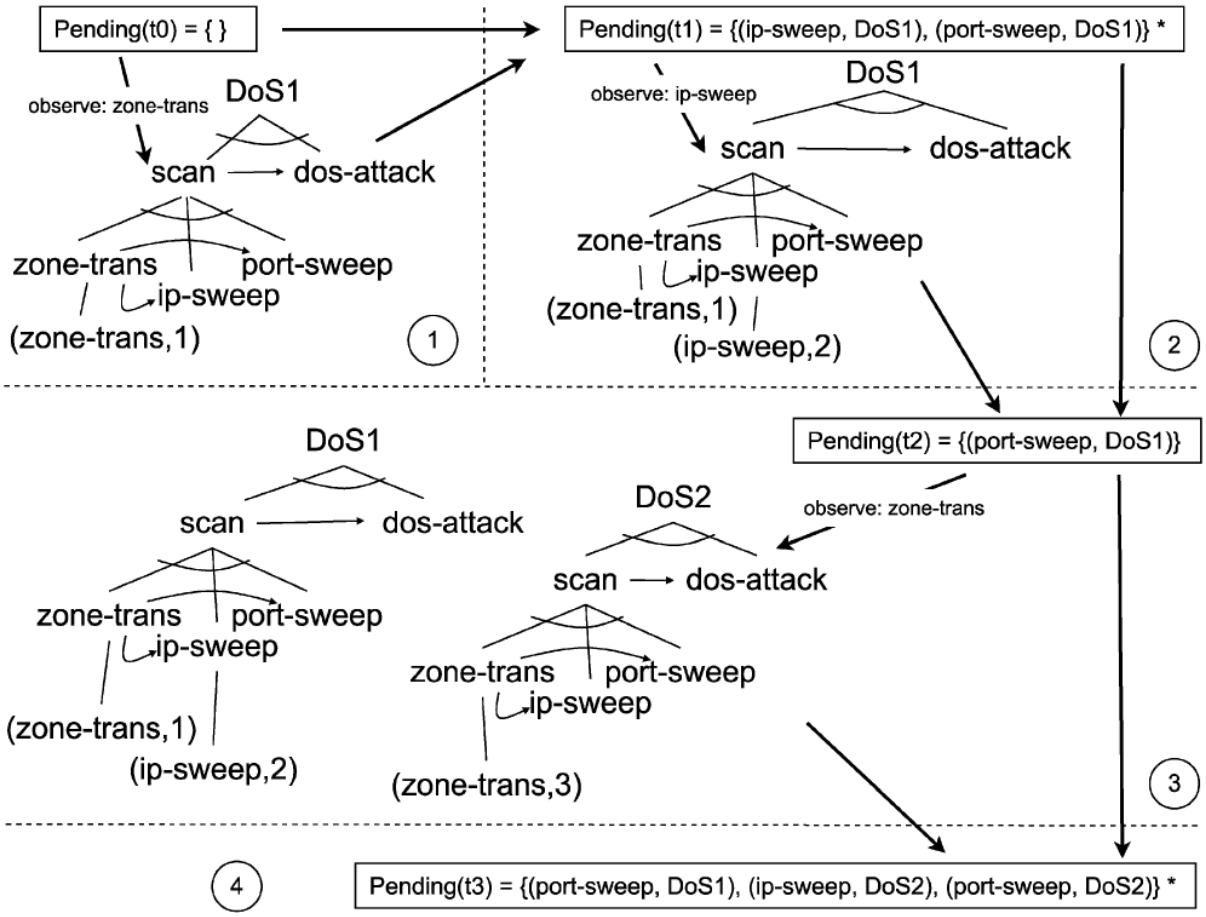


FIGURE 2 – Illustration des plans depuis l'article [Geib and Goldman, 2009]. Il est ici question d'une attaque DoS composée de deux noeuds ET imbriqués : DoS1 qui est composé de l'action scan puis dos-attack, et scan qui est composé d'abord de zone-trans, puis de ip-sweep et port-sweep (peu importe l'ordre). L'action zone-trans est réalisée en premier ce qui permet ce plan et donc on s'attend à observer les actions ip-sweep ou port-sweep. En deuxième, l'action ip-sweep est réalisée, le pending set est encore mis à jour et ne comporte plus que port-sweep. L'action zone-trans est alors répétée, PHATT instancie un deuxième plan d'une attaque DoS : DoS2. Le pending set est alors mis à jour en conséquence.

Deux algorithmes de lecture des arbres sont suggérés par Geib et Goldman : le premier est un algorithme descendant (*top-down*) qui permet de générer des explications et de les tester vis-à-vis des observations. Pour cela, trois probabilités sont prises en compte : la probabilité des buts à priori (à partir de leur fréquence), le choix d'une méthode plutôt qu'une autres (dont on suppose l'équi-

probabilité), et celle de choisir une action parmi toutes celles disponibles dans le pending set.

FIGURE 3 – Exemple de grammaire associée à ce type de plan [Geib and Goldman, 2009].

Theft → **scan get-ctrl get-data**: $\{(1,2)(2,3)\}$
scan → **zone-trans ip-sweep port-sweep**: $\{(1,2)(1,3)\}$
get-ctrl → **get-ctrl-local**: {}
get-ctrl → **get-ctrl-remote**: {}
get-data → **sniffer-install default-login**: {}

Le deuxième est ascendant (*bottom-up*) et consiste en un algorithme de programmation dynamique qui permet de construire les explications grâce aux séquences d'observations qu'il maintient à jour. Les observations attendues des ensembles de dérivations précédentes sont corrigées en fonction des observations qui suivent, et pour toute séquence d'observations, l'algorithme construit une explication partielle.

Dans les exemples illustrés aux figures 2 et 3, l'ordre est presque total (ip-sweep et port-sweep sont non-ordonnés) et l'on s'attend à ce que le hacker agisse dans un ordre précis. Cependant, dans des cas où il n'y a aucun ordre, nous sommes confrontés à une explosion combinatoire d'actions possibles. En effet, chaque nouvelle action peut soit continuer un plan déjà considéré, soit débuter une nouvelle instance. Un autre défaut est qu'il faut coder en dur les plans dans une bibliothèque. Ces inconvénient considérés, c'est en partant de PHATT qu'est créé le système HICOR (Hostile Intent Capability and Opportunity Recognizer) [Kabanza et al., 2010]. Ce système est appliqué dans le domaine du jeu vidéo, plus spécifiquement des jeux RTS (Real-Time Startegy). On peut représenter le jeu comme un environnement partiellement observable par un des joueurs.

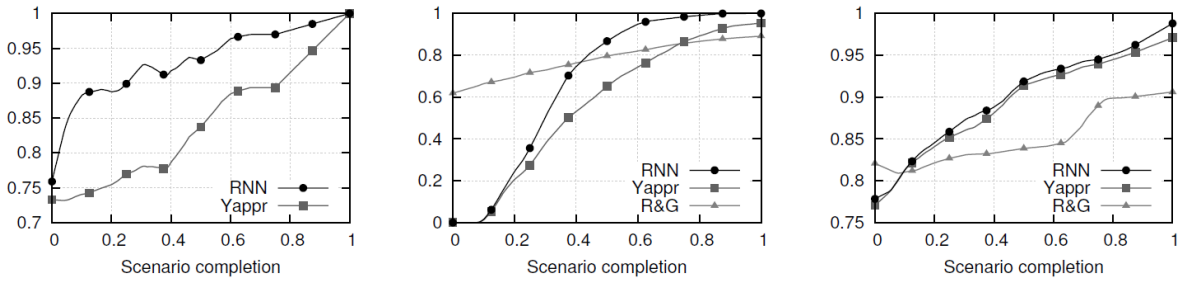
Les plans sont structurés en HTN (Hierarchical Task Networks), et la reconnaissance de ses plans est modulée comme étant un HMM (Hidden Markov Model). On ajoute des contraintes de temps sur la précédence entre deux événements pour apporter des précisions aux plans de la bibliothèque. L'algorithme de HICOR met à jour ses PES (Plan Execution Status) lorsque de nouvelles observations sont faites. Un PES est un tuple (T, (En0, . . . , Enn)) où T est une forêt d'arbres de planification partiellement complétés, et les E sont les actions possibles générées à chaque nouvelle observation (le pending set). Des contraintes de temps sont aussi ajoutées en plus de l'ordre. Pour calculer la probabilité de chaque PES, on prend en compte la probabilité à priori de chaque arbre de planification, puis la probabilité de chaque sous-arbre à un nœud OU dans l'arbre de planification, et enfin la probabilité que l'action e soit la prochaine action sélectionnée par l'agent étant donné une séquence d'actions précédentes. Les expérimentations sont ensuite faites sur des parties jouées par des joueurs professionnels.

2.2 Modélisation des plans grâce aux réseaux de neurones

Dans l'approche de PHATT qui utilise une bibliothèque de plans, le problème pour reconnaître le plan que choisit l'agent parmi tous les plans possibles est qu'il faut spécifier tous les plans dans la bibliothèque. Plutôt que d'avoir une bibliothèque codée en dur, l'approche de [Bisson et al., 2015] permet plus de flexibilité dans la reconnaissance de plan en vectorisant les plans de manière à les

apprendre avec un réseau de neurones récurrent. Pour cela, des hypothèses sont générées qui sont correctes ou incorrectes, et obtiennent alors un score qui permet de les ordonner et de sélectionner la première comme étant la plus probable. Le RNN (Recursive Neural Network) apprend directement les HTN de l'observé en fonction de la manière dont il atteint ses buts, et peut donc ensuite prédire ses actions.

Les expérimentations sont faites sur des exercices de références où l'algorithme de [Bisson et al., 2015] est comparé à d'autres algorithmes à la pointe de la reconnaissance de plan. Le "Monroe plan corpus" traite de la planification d'une intelligence artificielle en gestion de crise, le deuxième est "StarCraft Navigation"(SCN) et traite de la reconnaissance de plan au sein de StarCraft, et le dernier est "Kitchen" où différentes actions sont effectuées dans une cuisine et où il faut identifier ce qui y est préparé. Dans les trois cas, le RNN converge plus rapidement vers le but exécuté.



(a) Monroe plan corpus (b) StarCraft navigation (c) Kitchen

FIGURE 4 – Résultats des expérimentations tels que présenté par [Bisson et al., 2015]. On observe que le RNN converge plus rapidement vers le bon but, et prédit donc mieux avec moins d'informations que les autres.

2.3 La planification inverse : une méthode de reconnaissance de plan

La reconnaissance de plan consiste à observer des actions et en déduire les buts, alors que la planification déduit les actions à réaliser à partir du but fixé. Cette nouvelle approche consiste à se mettre à la place de l'agent pour trouver un plan optimal qui satisfasse son objectif. On ne cherche plus à prédire les prochaines actions mais seulement le but. L'avantage de cette approche est qu'elle s'affranchit de l'utilisation d'une librairie de plans [Ramırez and Geffner, 2009]. À la place, on cherche la séquence d'actions optimales (à moindre coût) pour atteindre un but en travaillant sur les actions disponibles. Le coût du plan pour un but donné est calculé, puis est comparé au coût pour ce même but mais en prenant en compte les observations de l'agent observé. Nous sommes dans un contexte où l'environnement est entièrement observable. Dans le cas d'une égalité, il est alors possible que l'agent suive bien ce but, sinon ce but est filtré et est retiré des buts possibles. Les problèmes de cette méthode sont qu'elle filtre les plans mais ne leur assigne pas de probabilités ni de pondération d'aucune sorte ; elle ne permet de suivre qu'un seul but, et s'adapte mal au bruit introduit par des agents suivants des plans sous-optimaux. Pour régler le problème du filtrage, et passer à une distribution probabiliste sur les différents buts, on peut utiliser la distance entre le coût optimal du plan et le coût en fonction des observations.

Une approche similaire est proposée, mais qui assigne des probabilités aux différents buts. Cette nouvelle méthode permet de s'appliquer à des comportements qui ne sont pas optimaux et est donc plus résistante au bruit induit dans les observations [Ramırez and Geffner, 2010]. Pour cela, on se base sur la distance entre le coût pour atteindre un but en exécutant les même actions réalisés par l'agent observé (comme dans la méthode précédente), et le coût minimal pour ce même but mais sans pouvoir exécuter les actions observées. Cela induit la pénalité subite en cherchant à accomplir le même but sans passer par les actions de l'agent. On note c(G,O) le coût de réalisation du but G en exécutant les observations O.

$$\Delta(G, O) = c(G, O) - c(G, \neg O)$$

En passant par une distribution de Boltzman, on obtient :

$$\frac{P(O|G)}{P(\neg O|G)} = e^{-\beta\Delta(G,O)}$$

De cette équation, de la règle de Bayes et des probabilités des buts à priori (considérés comme équiprobable dans les expérimentations), nous déduisons P(G|O). Alors que l'élagage ne laissait qu'une faible tolérance aux comportements non optimaux des agents observés, cette nouvelle approche règle ce problème en diminuant seulement la probabilité des buts plutôt que de les éliminer totalement.

L'intérêt de cette approche est qu'elle permet d'implémenter facilement les meilleurs algorithmes d'exploration de graphe et donc s'adapte facilement à de lourds problèmes en terme de complexité. Les performances de l'algorithme ont été testées en utilisant 3 classificateurs différents : HSP_F^*, anytime LAMA, et greedy LAMA. On observe que HSP_F^* prend plus de temps en moyenne que anytime LAMA pour calculer les probabilités des plans (jusqu'à 10^3 fois plus long, variant en fonction des expériences), qui lui aussi est bien plus long (en moyenne 500% plus long) que greedy LAMA, cependant il n'y a que une faible perte en qualité entre anytime LAMA et greedy LAMA, et HSP_F^* et anytime LAMA sont presque équivalente.

La planification inverse peut aussi être utilisée dans un contexte plus général, qui ne nécessite pas de supposer que les actions de l'agent sont déterministes et que celui-ci a une observabilité complète de son environnement. On passe alors à un problème de reconnaissance de but dans un POMDP (Partially Observable Markov Decision Process) [Ramırez and Geffner, 2011].

Ici, le concept d'observation partielle n'est pas le même pour l'observé et pour l'observateur. L'observateur n'observe pas toutes les transitions d'un état à l'autre, c'est-à-dire toutes les actions de l'agent. En revanche, l'agent observé est toujours conscient de toutes les actions qu'il accomplit, mais il ne peut pas observer tous les aspects de l'état dans lequel il se trouve et par conséquent il ne peut savoir avec certitude dans lequel il se trouve. C'est pourquoi la notion d'état de croyance est introduite. Ces états de croyances permettent de prendre en compte la probabilité d'être dans un état plutôt que dans un autre, et de simuler un système complètement observable. Une fois ramené à un modèle markovien, il est expérimenté en fournissant différents niveaux de complétude de la séquence d'observations pour trouver la distribution de probabilité des différents buts (30 %, 50%, et 70% de complétion des observations). Les performances comme on pouvait s'y attendre s'améliorent avec le niveau de complétion, obtenant des résultats presque optimaux à 70%.

2.4 L'apprentissage profond pour reconnaître les buts dans un jeu à monde ouvert

Les mondes ouverts dans les jeux vidéo donnent aux joueurs une liberté qu'ils apprécient beaucoup, mais qui posent de nombreuses difficultés aux développeurs pour garder l'ensemble cohérent et qui nécessitent souvent beaucoup de règles pour garder un mélange homogène d'événements, allant avec la trame scénaristique. Ce type de reconnaissance de plan est partiellement résolu grâce à des méthodes innovantes d'apprentissage automatique dans ce domaine [Min et al., 2014].

Nous sommes donc dans un contexte où l'agent observé peut poursuivre un plan parmi d'autres de plusieurs manières différentes. Les buts du jeu sont déduits par les joueurs et les buts que tentent d'atteindre les joueurs sont déduits de par les événements qu'ils déclenchent et les actions qu'ils réalisent.

Il est pris comme postulat que chaque action est réalisée dans la poursuite d'un unique but, et qu'une action ne sert pas à deux buts différents. Leurs expérimentations sont faites sur un jeu éducatif appelé CRYSTAL ISLAND où le joueur doit découvrir la mystérieuse cause d'une infection. Les actions du joueur vont servir comme paramètres en entrée du réseau de neurones et prennent en compte : l'emplacement du joueur, les buts atteints précédemment, l'état d'avancement de l'histoire, et le type d'actions qu'il réalise. Un deuxième ensemble d'actions identiques prend en plus compte de l'argument de cette action.

Pour l'apprentissage, ils prennent seulement en compte les n derniers buts observés pour prédire le but recherché. Afin de rendre le réseau de neurones plus robuste au bruit, des auto-encodeurs de débruitage sont implémentés pour réduire l'impact des actions non pertinentes (le bruit) et éviter un apprentissage trivial des poids. Différents réseaux de neurones sont donc entraînés en faisant varier le nombre et le type de caractéristiques données en entrée au réseau, mais aussi le nombre de buts précédents considérés, le nombres de couches cachées ainsi que le nombre d'époques réalisées. Le modèle gardé pour l'analyse des résultats est celui avec le taux d'erreur le plus faible. Des comparaisons sur la précision, vitesse et point de convergence vis-à-vis des buts sont faites avec une autre étude réalisée plus tôt [Ha et al., 2011]. Un résultat intéressant est que l'ensemble comprenant les paramètres des actions produit de moins bon résultats que celui sans, indiquant que les données supplémentaires étaient trop clairsemées pour apporter de l'information pertinente.

2.5 La reconnaissance d'intention avec AlphaGo

Le jeu de GO est un un ancien jeu chinois de contrôle de territoire. Il se joue en un contre un en posant tour à tour une pierre de sa couleur sur les intersections d'un tablier quadrillé appelé goban de taille 19*19. C'était jusqu'en mars 2016 le dernier grand jeu classique de stratégie (comme les échecs ou les dames) où les IA ne pouvaient pas battre les meilleurs humains dû à sa richesse combinatoire. Cependant Silver et al. ont créé une IA capable de battre les meilleurs joueurs humains et les autres IA jouant à Go, qu'ils ont appelé AlphaGo [Silver et al., 2016]. Dû au grand nombre de possibilités exponentiel de déroulement d'une partie, il est impossible de rechercher tous les coups possibles, car il y a en moyenne 250 coups possibles par tour et la partie se joue en 150 coups (combiné des deux joueurs) environ, soit environ 250^{150}différentes parties. Il est donc impossible de calculer tous les aboutissements possibles malgré la puissance formidable déployée par Google DeepMind où il a nécessité un total de 1 202 CPUs et 176 GPUs pour accomplir l'exploit de battre le champion du monde en titre Lee Sedol. Cette puissance de calcul était exploitée par

plusieurs ingénieux réseaux de neurones travaillant de concert pour choisir le meilleur coup possible.

Le “policy network” (réseau de politiques) permet de choisir le prochain coup. Ce réseau de neurones a une profondeur de 13 couches et commence par une phase d’apprentissage supervisée (SL-Supervised Learning) où il est question d’apprendre les coups faits lors de parties de joueurs experts. À la fin de cette phase, il est capable de prédire le prochain coup d’un joueur professionnel avec une précision de 55,7% comparée à 44.4% le maximum précédent établi par d’autres équipes de chercheurs.

Une version beaucoup plus rapide mais moins performante (24% de précision) est établie en prenant en compte une fenêtre plus petite centrée autour des coups précédents, nommée “rollout policy” (politique de déploiement). Le temps moyen pour choisir le prochain coup passe alors de 3ms à 2 µs dans cette version.

C’est à partir du "policy network" qu’est créée la version renforcée (RL-Reinforced Learning) où AlphaGo va jouer contre une version précédente de lui-même, sélectionnée aléatoirement afin d’éviter le surapprentissage. Cette nouvelle version en mode RL gagne 80 % des parties jouées contre lui-même en mode SL.

Ensuite, le “value network” (réseau de valeur) est créé grâce à la simulation des parties d’AlphaGo (RL) jouées contre lui-même, afin de pouvoir évaluer les différentes variations d’un plateau de jeu. Le problème est qu’en prenant uniquement en compte le résultats des ces parties (gagnées ou perdues) cela mène au surapprentissage lorsque les parties sont issues de celles jouées par les joueurs. Néanmoins, ce problème est réglé en générant 30 millions de nouvelles parties d’AlphaGo jouées contre lui-même.

Enfin est implémenté la recherche arborescente de Monte-Carlo (MCTS) qui permet de choisir le meilleur coup possible parmi ceux disponibles en exécutant plusieurs simulations. Pour ce faire, l’algorithme combine le policy network et le value network de la façon suivante : lors des simulations, une combinaison du policy network (SL) et de la valeur résultante des simulations précédentes de Monte Carlo décide du prochain coup à exécuter. À la fin d’une simulation, la feuille de l’arbre est évaluée en combinant la valeur retournée par le « value network » (RL) avec le résultat d’une partie jouée avec la «rollout policy», ce qui permet de simuler plus rapidement des parties du début à la fin.

Dans ce contexte, nous ne cherchons plus à inférer le but final de notre adversaire qui est connu dès le début, à savoir contrôler le plus de territoire possible et capturer le plus possible de prisonniers. AlphaGo comme la plupart des IA dans un contexte similaire, utilise les mêmes méthodes pour déterminer son coup et celui de l’adversaire, cela induit qu’il est pris comme hypothèse que son adversaire raisonne de la même manière. C’est pourquoi il est utile d’avoir entraîné le réseau de neurones SL en se basant sur des joueurs. Il serait par contre intéressant d’utiliser une approche s’inspirant d’AlphaGo pour faire la reconnaissance de plan dans un contexte d’adversité. Un effort d’abstraction est fait par le joueur afin de prédire les coups de son adversaire en fonction des siens, mais il n’y a pas de plans bien précis établi depuis le début. Les méthodes utilisées permettent de prédire des stratégies qui suivent certains schémas mais cela reste très différent de l’usage classique des bibliothèques de plans.

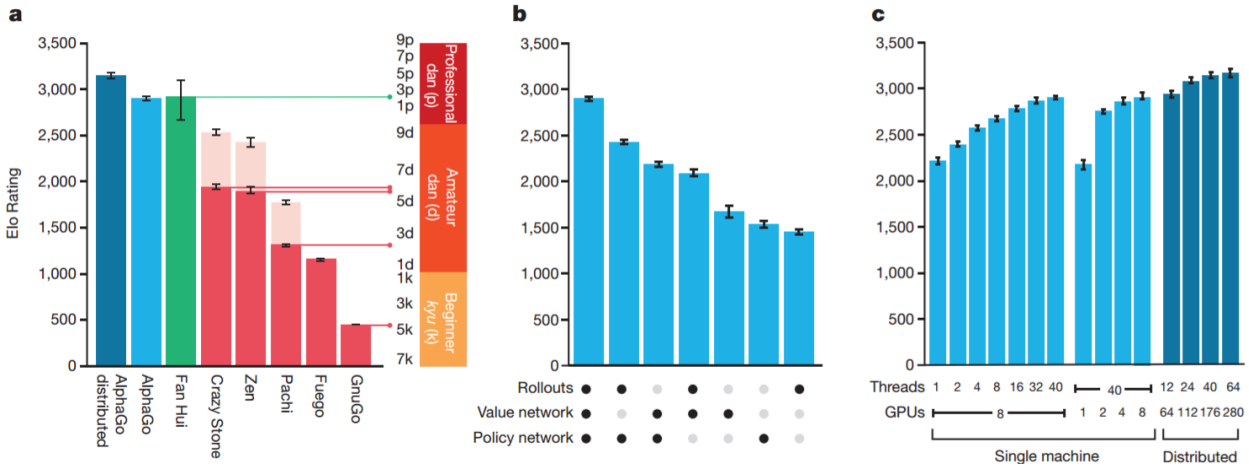


FIGURE 5 – Résultats des différentes expérimentations tels que présentés par [Silver et al., 2016]. Dans le premier histogramme (a) nous avons un comparatif des différentes IA du jeu de GO en fonction de leur Elo. L'Elo est une mesure classique pour estimer la valeur d'un joueur dans un jeu à deux joueurs comme le tennis ou les échecs. On fait ici la distinction entre AlphaGo tournant sur un seul ordinateur et la version distribuée utilisant des CPUs et GPUs travaillant en parallèle. Le graphique (b) montre comment les performances des versions d'AlphaGo évoluent en fonction des modules qu'il utilise. Il est le plus performant lorsqu'il combine les trois pour décider ses coups avec plus de marge qu'entre n'importe quelle combinaison. Enfin en (c), nous observons qu'AlphaGo, dans sa version distribuée comparativement à sur une machine unique, gagne en performance en multipliant le nombre de threads qu'elle utilise pour ses calculs.

3 Conclusion

Dans cette bibliographie, nous avons pu voir les différentes approches faisant états-de-l'art dans le domaine de la reconnaissance de plan. Certaines approches seront expérimentées lors du stage au laboratoire Planiart de l'Université de Sherbrooke, comme celle de [Ramırez and Geffner, 2010], où ils créent une distribution probabiliste sur différents buts en utilisant des planificateurs classique dans leur algorithme de planification inverse. Il pourrait être intéressant de combiner cette méthode de planification inverse avec celle plus rapide des HTN. On essaiera également, par la suite de généraliser dans un contexte stochastique en utilisant des MDPs (Markov Décision Process) comme présenté dans [Song et al., 2013] ou en s'inspirant du travail réalisé avec les POMDPs. Le but final est de le généraliser dans un jeu à 2 adversaires et de tester différents types de techniques d'apprentissage automatique pour améliorer les performances.

Références

[Aumann, 1989] Aumann, R. J. (1989). Game theory. In *Game Theory*, pages 1–53. Springer.

[Bisson et al., 2015] Bisson, F., Larochelle, H., and Kabanza, F. (2015). Using a recursive neural network to learn an agent's decision model for plan recognition. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 918–924.

[Cohen et al., 1981] Cohen, P. R., Perrault, C. R., and Allen, J. F. (1981). Beyond question answering. *Strategies for natural language processing*, pages 245–274.

[Geib and Goldman, 2009] Geib, C. W. and Goldman, R. P. (2009). A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 173(11) :1101–1132.

[Gold, 2010] Gold, K. (2010). Training goal recognition online from low-level inputs in an action-adventure game. In *AIIDE*.

[Ha et al., 2011] Ha, E., Rowe, J. P., Mott, B. W., and Lester, J. C. (2011). Goal recognition with markov logic networks for player-adaptive games. In *AIIDE*.

[Kabanza et al., 2010] Kabanza, F., Bellefeuille, P., Bisson, F., Benaskeur, A. R., and Irandoust, H. (2010). Opponent behaviour recognition for real-time strategy games. *Plan, Activity, and Intent Recognition*, 10 :05.

[Min et al., 2014] Min, W., Ha, E., Rowe, J. P., Mott, B. W., and Lester, J. C. (2014). Deep learning-based goal recognition in open-ended digital games. In *AIIDE*. Citeseer.

[Pentney et al., 2006] Pentney, W., Popescu, A.-M., Wang, S., Kautz, H., and Philipose, M. (2006). Sensor-based understanding of daily life via large-scale use of common sense. In *AAAI*, volume 2, page 2.

[Ramırez and Geffner, 2009] Ramırez, M. and Geffner, H. (2009). Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc*, pages 1778–1783.

[Ramırez and Geffner, 2010] Ramırez, M. and Geffner, H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*, pages 1121–1126. Citeseer.

[Ramırez and Geffner, 2011] Ramırez, M. and Geffner, H. (2011). Goal recognition over pomdps : Inferring the intention of a pomdp agent. In *IJCAI*, pages 2009–2014. IJCAI/AAAI.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587) :484–489.

[Song et al., 2013] Song, Y. C., Kautz, H., Allen, J., Swift, M., Li, Y., Luo, J., and Zhang, C. (2013). A markov logic framework for recognizing complex events from multimodal data. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 141–148. ACM.