UNE COMPARAISON D'ALGORITHMES DE RECONNAISSANCE DE PLAN À L'AIDE D'OBSERVATIONS *IN SITU*

par

Cody Stoutenburg Tardieu

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 22 juillet 2015

Le 22 juillet 2015

le jury a accepté le mémoire de Monsieur Cody Stoutenburg Tardieu dans sa version finale.

Membres du jury

Professeur Froduald Kabanza
Directeur de recherche
Département d'informatique

Professeur Sylvain Giroux
Codirecteur de recherche
Département d'informatique

Professeur Gabriel Girard
Membre interne
Département d'informatique

Professeur Richard St-Denis
Président-rapporteur
Département d'informatique

Sommaire

Ce mémoire présente une comparaison de deux algorithmes de reconnaissance de plan, soit YAPPR (Yet Another Probabilistic Plan Recognizer) et PR-Plan (Plan Recognizer as Planning). Afin de comparer les algorithmes, nous avons voulu utiliser un domaine plus complexe et réaliste que ceux utilisés jusqu'à présent. Pour ce faire, nous avons établi un protocole de comparaison en utilisant le concept d'observation *in situ*. Nous avons utilisé le jeu de stratégie en temps réel StarCraft comme environnement de simulation. Puis, nous avons créé un agent jouant à StarCraft qui utilise la reconnaissance de plan comme élément central pour le système de prise de décision. Pour valider que notre principe d'observation *in situ* fonctionne, nous avons créé des agents témoins et exécuté de nombreuses simulations.

Mots-clés: Reconnaissance de plan ; Agent *in situ* ; Intelligence artificielle ; Jeux de stratégie en temps réel.

Sommaire

Remerciements

Je remercie mon directeur de recherche Froduald Kabanza pour avoir accepté de diriger mes travaux et pour son soutient financier. Je remercie aussi Sylvain Giroux d'avoir codirigé mes travaux. Leurs conseils pertinents, leurs encouragements et leur compréhension ont été précieux. Ils m'ont appris à canaliser mes idées et à être plus rigoureux dans mes travaux.

Je remercie mes grands-parents pour m'avoir toujours incité à lire, ma mère pour son écoute toujours attentive et mes amis pour leurs encouragements.

Merci à ma femme et ma fille, qui illuminent chacune de mes journées, pour leur patience et leur soutien.

Remerciements

Abréviations

PHATT Probabilistic Hostile Agent Task Traker

YAPPR Yet Another Probabilistic Plan Recognizer

DOPLAR Decision-Oriented PLAn Recognizer

PROBE Provocation for the Recognition of Opponent BEhaviours

PR-Plan Plan Recognizer as Planning

HTN Hierarchical Task Networks (Réseaux de tâches hiérarchiques)

Table des matières

Liste des figures

Liste des tableaux

Liste des programmes

Introduction

Mise en contexte

L'intelligence artificielle est un vaste domaine dont un des buts est de créer ou simuler un processus cognitif similaire à celui de l'humain. Un des problèmes fondamentaux étudiés en intelligence artificielle est la *reconnaissance de plan*, c'est-à-dire la capacité cognitive de reconnaître les intentions d'un autre agent. Ce processus est couramment utilisé, par exemple, dans le sport quand on cherche à anticiper ce que vont faire les adversaires ou les coéquipiers. Parfois, on cherche uniquement à connaître l'objectif de l'autre agent, par exemple connaître la cible d'une attaque. D'autres fois, on veut découvrir le plan exact. Dans le cadre de ce mémoire, nous portons notre attention sur l'analyse de ce processus afin de comparer différents algorithmes de reconnaissance de plan.

Description du problème

Dans cette section, nous définissons ce qu'est la reconnaissance de plan, les différentes approches et les motivations pour résoudre ce problème.
Le problème de reconnaissance de plan est décrit pour la première fois par Schmidt *et al.* [18] en 1978. La reconnaissance de plan se définit comme le fait de vouloir déterminer les intentions d'individus ou d'agents. Il est possible de résoudre ce problème en analysant les actions qu'effectue l'agent dans son environnement. Cela nécessite la présence d'au moins deux intervenants, l'observateur et l'observé. L'interaction entre les deux intervenants sera décrite plus tard. On peut donc voir la reconnaissance de plan comme un problème où il faut inférer le plan de l'agent

observé considérant un ensemble d'observations.

La résolution de ce problème est complexe et différentes approches existent. Ici, on distingue deux approches principales : d'une part, les approches non génératives, qui nécessitent une connaissance *a priori* de l'ensemble des plans que l'on veut reconnaître ; d'autre part, les approches génératives, qui ne nécessitent qu'une description de l'environnement et de l'ensemble des objectifs potentiels.

La capacité de reconnaître les intentions d'un agent est très utile dans divers domaines. En effet, en domotique il est important de savoir ce que fait l'habitant d'un appartement intelligent afin de pouvoir l'assister. Cela est très utile aussi pour de la surveillance afin de détecter des activités suspectes. La reconnaissance de plan peut être très utile dans le domaine militaire afin de pouvoir comprendre ce que fait l'ennemi ; cela s'applique aussi pour les jeux de stratégie. Cependant dans l'ensemble de ces situations, le ou les agents observés ne vont pas se comporter de la même façon vis-à-vis de leur observateur et la qualité des observations risque d'être différente selon la situation.

Les types d'agents observés

Nous détaillons les différents types d'agents qu'il est possible de rencontrer selon le domaine étudié. Il est possible de rencontrer trois types d'agents différents, l'agent collaboratif, indifférent ou hostile.

Un agent collaboratif a conscience d'être observé et il va agir de telle sorte que ses actions vont nous aider à reconnaître son plan. Il veut nous assister dans la reconnaissance de son plan. On retrouve un tel agent dans un jeu coopératif. Si je ne peux pas communiquer avec un agent qui est mon coéquipier, celui-ci va agir de façon à ce que je puisse rapidement comprendre ce qu'il fait. On observe ce type de comportement dans les sports d'équipes. Par exemple, le football où les joueurs ne peuvent pas communiquer à cause du bruit, des adversaires et de la distance.

À l'opposé, un agent hostile a lui aussi conscience d'être observé, par contre il ne désire pas que l'on sache ce qu'il fait. Non seulement il ne va pas nous aider à reconnaître son objectif, mais il risque en plus de le masquer en faisant des actions qui vont nous induire en erreur. Il va introduire du bruit et de la tromperie dans son

comportement. On retrouve un tel agent dans un jeu adversariel. Notre adversaire va tenter de prendre l'avantage sur nous et ce faisant il va masquer sa stratégie et ses objectifs afin de ne pas être contré.

Enfin, un agent indifférent est un agent qui n'a pas conscience qu'on l'observe ou que cela indiffère. Cela implique que l'agent ne va pas tenir compte du fait qu'on l'observe et va garder son comportement normal. Par exemple, une personne qui vit dans une maison intelligente ne va plus tenir compte du fait qu'il soit observé. Il ne va donc pas masquer ses intentions, par contre il est possible qu'il fasse des erreurs, utilise des plans sous-optimaux, mais pas dans l'objectif de tromper son observateur.

Les différentes qualités d'observation

Nous avons vu qu'il existe différents agents dont le comportement change selon leur attitude vis-à-vis de l'observateur. La capacité d'observation de l'observateur joue aussi un rôle important dans le problème. Cette capacité peut dépendre beaucoup de l'environnement, mais aussi des capacités d'observation intrinsèques de l'observateur (ses capteurs et capacités de traitement des observations). Dans cette optique, nous allons considérer des observateurs avec une capacité d'observation parfaite et ceux avec une capacité d'observation partielle. Sans perte de généralité, nous pouvons considérer que la capacité d'observation est une propriété de l'environnement ou du domaine, et nous parlons alors d'environnement ou domaine parfaitement observable ou partiellement observable.

L'observabilité parfaite correspond à la situation où l'observateur voit tout avec précision. Il observe chacune des actions de l'agent observé sans aucun bruit et avec certitude. Cet environnement est peu fréquent et est souvent limité aux simulations. Il est aussi utilisé comme hypothèse pour simplifier la résolution des problèmes de reconnaissance de plan.

L'observabilité partielle se définit de deux façons. Il est possible que certaines zones de l'environnement ne soient pas observées. Cela signifie qu'il est possible que l'on ne perçoive pas l'ensemble des actions de l'agent observé. Il est possible que certains capteurs de l'observateur soient défectueux. Cela peut introduire du bruit et des observations manquantes mal interprétées. La plupart des environnements ont une

observabilité partielle pour l'une des deux raisons décrites, voire même les deux.

Comparer des algorithmes de reconnaissance de plan

La comparaison d'algorithmes de reconnaissance de plan n'est pas aussi évidente que cela pourrait sembler. Premièrement, les algorithmes sont souvent testés avec des jeux de données créés pour l'algorithme lui-même. Deuxièmement, les quelques jeux de données partagées par les chercheurs sont souvent trop simples et ne permettent pas de tirer des conclusions fiables sur l'efficacité des algorithmes en pratique. Pour cette raison, nous avons cherché un domaine plus complexe. Nous avons décidé de faire notre comparaison sur un jeu de stratégie en temps réel (StarCraft). Ce domaine est très complexe et permet la reconnaissance de plan avec un agent hostile. Bien que nous ayons ciblé le domaine précis de StarCraft, nous croyons que l'approche pourrait être généralisée à d'autres domaines. Par exemple, pour la reconnaissance des activités de la vie quotidienne, il serait possible de reprendre notre cadre de comparaison avec un agent indifférent. Pour ce faire, il faudrait faire la modélisation du domaine pour les différents algorithmes ce qui n'est pas une tâche facile.

Organisation du mémoire

Le mémoire est découpé en cinq chapitres. Nous allons d'abord décrire l'état de l'art dans la reconnaissance de plan. Ensuite, nous présentons StarCraft, le domaine utilisé pour notre comparaison *in situ*. Nous détaillons les scénarios créés pour nos simulations. Ensuite, nous expliquons l'architecture et le fonctionnement de notre agent. Après cela nous analysons les résultats obtenus lors de nos simulations. Finalement, nous présentons des pistes pour de futurs travaux de recherche.

Chapitre 1

État de l'art

Dans ce chapitre, nous présentons les principales approches qui existent dans le domaine de la reconnaissance de plan. Puis nous décrivons les principaux algorithmes qui représentent l'état de l'art. Nous expliquons aussi le choix des algorithmes et du domaine pour notre comparaison.

1.1 Les approches

1.1.1 Approche symbolique

L'approche symbolique fut l'une des premières approches mises de l'avant pour faire de la reconnaissance de plan par Schmidt *et al.* [18], Kautz et Allen [11]. Elle considère que l'on peut définir une taxonomie d'actions et ainsi le plan reconnu serait celui dans lequel le plus d'actions de plus bas niveau sont reconnues. Cela se fait en utilisant le principe du rasoir d'Occam, c'est-à-dire en considérant qu'entre deux plans qui accomplissent le même but, le plus simple est le plus vraisemblable. Cette approche tire des conclusions hâtives, car si deux plans semblent probables elle considère systématiquement le plus simple comme étant le plus probable. De plus, cela ne tient pas compte du fait que l'agent peut faire des actions pour nous tromper ou par erreur.

1.1.2 Approche probabiliste

Le concept de l'approche probabiliste est le calcul des probabilités des buts sachant les observations. Une fois les probabilités calculées, les hypothèses peuvent être ordonnancées. L'approche probabiliste nécessite la connaissance de la probabilité *a priori* des buts possibles de l'agent observé. Un modèle possible proposé par Charniak et Goldman [3] procède par la création d'un réseau bayésien comportant l'ensemble des actions possibles et des objectifs à atteindre. Cette approche est très puissante, car elle permet de prendre en considération le comportement d'un agent, qu'il soit amical, neutre ou même hostile. En effet, il est possible de modéliser le fait qu'un agent essaie de tromper l'observateur. Il suffit pour cela d'ajuster les liens entre les actions et de connaître les tactiques de duperie potentiellement utilisées par l'agent observé. Par contre, la grande difficulté de cette approche réside dans la création et le calcul *a priori* des probabilités entre chacune des actions et des plans. En effet, il est nécessaire de connaître l'ensemble des actions possibles avec la distribution de probabilités les caractérisant. De plus, la distribution de probabilités est en pratique difficile à spécifier.

1.1.3 Approche hybride

Les approches symboliques et probabilistes ayant chacune des limitations, plusieurs travaux récents s'appuient sur des approches hybrides qui utilisent une taxonomie qui va être appuyée par un modèle probabiliste. Cela permet d'utiliser des probabilités afin de modéliser des schémas plus complexes à la place d'utiliser uniquement le principe du rasoir d'Occam. Ces approches requièrent des spécifications des distributions de probabilités plus simples comparativement aux approches utilisant un réseau bayésien. Geib et Goldman ont fait une implémentation hybride utilisant le concept de bibliothèque de plans, PHATT (*Probalistic Hostile Agent Task Traker*) [9], qui fut améliorée par la suite avec YAPPR (*Yet Another Probabilistic Plan Recognizer*) [7] en s'appuyant sur le concept de grammaire.

Cette approche est une des approches hybrides les plus reconnues. Plusieurs travaux tendent à améliorer le concept. Geib et Goldman ont modifié l'algorithme pour gérer

les plans circulaires [8]. Par la suite, Bisson l'a adaptée pour utiliser la provocation de l'adversaire pour réduire l'ambiguïté du plan reconnu avec PROBE (Provocation for the Recognition of Opponent BEhaviours) [1]. Kabanza *et al.* ont fait un nouvel algorithme, DOPLAR (Decision-Oriented PLAn Recognizer) [10], sur les bases de YAPPR, et qui réussit à calculer des probabilités strictement bornées et ainsi à améliorer le temps de calcul des probabilités.

1.1.4 Approche générative

L'approche générative considère qu'il n'est pas raisonnable de vouloir spécifier l'ensemble des plans possibles de l'agent observé *a priori*. Au contraire, cette approche cherche à définir l'environnement, les actions primitives de l'agent observé et ses buts potentiels. On transforme alors le problème de la reconnaissance de plan en un problème de prise de décision inversé. La difficulté de ces approches réside dans la définition d'algorithmes de prise de décision efficaces. Par exemple, Ramírez et Geffner ont introduit l'algorithme PR-Plan (Plan Recognizer as Planning) qui a recourt à des algorithmes de planification classiques [15]. Braynov [2] fait un lien entre la planification adversariel et la reconnaissance de plan.

1.1.5 Autres approches

Il existe aussi bien d'autres approches, y compris certains travaux qui utilisent des algorithmes d'apprentissage. Par exemple, Weber et Mateas [19] utilise le forage de données pour déterminer la stratégie de l'adversaire dans le jeu StarCraft. La faiblesse de ces approches est le manque de généralité de la solution. En effet, ils se limitent à la résolution d'un problème spécifique, mais pas à l'ensemble de la reconnaissance de plan. D'autres approches considèrent l'utilisation de la théorie des jeux pour la reconnaissance de plan [14].

1.2 Les algorithmes comparés

Dans le reste de cette revue de littérature, nous décrivons plus en détail les algorithmes que nous avons comparés dans ce travail, c'est-à-dire YAPPR et

PR-Plan. Avant de donner cette description, il convient de souligner que ce projet va bien au-delà d'une comparaison traditionnelle. Il introduit en plus un nouveau cadre de comparaison d'algorithmes *in situ*, utile lorsqu'on ne dispose pas d'un jeu de données étiquetées.

1.2.1 Yet Another Probabilistic Plan Recognizer

YAPPR est un algorithme de reconnaissance de plan qui utilise le concept de bibliothèque de plans [7]. Chaque plan est défini sous forme d'un réseau de tâches hiérarchiques (*Hierarchical Task Networks*, HTN), représenté par une grammaire hors contexte partiellement ordonnée ou un graphe ET-OU partiellement ordonné. Cela permet de définir avec simplicité une grande quantité de plans. Il suffit de définir l'ensemble des actions primitives possibles puis de les ordonnancer en arbres pour obtenir un plan.

Pour chaque observation YAPPR va regarder l'ensemble des plans ou combinaisons de plans pour lesquelles l'action peut être expliquée. Cela représente un ensemble d'explications possibles. Si l'action ne peut pas être expliquée, le système de reconnaissance de plan échoue. C'est pourquoi il est important de construire une bibliothèque de plans exhaustive.

Puis pour chacune des explications possibles, il calcule une probabilité considérant plusieurs facteurs tels que la probabilité d'occurrence de l'observation, sachant que la probabilité d'occurrence de l'observation tient compte des observations précédentes et les probabilités *a priori* de chacun des plans. Ensuite, YAPPR maintient un ensemble ouvert de plans possibles avec une probabilité associée à chacun d'eux. Cette probabilité peut être bornée grâce à des améliorations apportées par DOPLAR [10]. Ainsi à mesure que des observations s'ajoutent, les probabilités vont s'ajuster et les explications invraisemblables vont être rejetées.

Les avantages de cette approche sont nombreux. Tout d'abord, comme nous modélisons la bibliothèque de plans cela permet de modéliser divers types d'agents (par exemple malade, adversariel, rationnel, *etc.*). L'approche facilite la reconnaissance de plans multiples et entrelacés. La vitesse d'exécution de cet algorithme est aussi un grand avantage. Il est capable de calculer rapidement les

probabilités des plans de l'agent, car il lui suffit de chercher dans un ensemble fini d'hypothèses. Ainsi il est capable de rejeter rapidement des plans invraisemblables.

Les faiblesses de l'approche sont aussi multiples. Tout d'abord la création d'une bibliothèque de plans n'est pas une tâche facile surtout que celle-ci doit être exhaustive sans quoi on échouera à déterminer le plan de l'agent. Ensuite, la nécessité de suivre exactement un plan rend l'algorithme très sensible au bruit ou à toute forme de duperie. Ensuite, le fonctionnement de l'algorithme veut qu'il conserve toutes les hypothèses de plans, ce qui l'empêche de détecter tout abandon de plan. Dans ce cas, il considère que l'agent suit un autre plan en plus des plans précédemment suivis.

1.2.2 Plan Recognizer as Planning

L'algorithme Plan Recognizer as Planning (PR-Plan) de Ramírez et Geffner utilise la planification afin de trouver les objectifs de l'agent [15]. PR-Plan nécessite une modélisation des actions primitives de l'agent ainsi que l'ensemble des buts possibles.

L'intuition de cet algorithme est que de façon générale, entre deux plans qui accomplissent le même but, le plan optimal est le plus probable, si on suppose que l'on a affaire à des agents relativement rationnels. Ainsi, étant donné une séquence d'observations des actions de l'agent, la probabilité qu'un but particulier soit celui poursuivi par un agent est proportionnelle à la différence entre le coût du plan optimal cohérent avec les observations et le coût du plan optimal non cohérent avec les observations.

PR-Plan, lors d'une nouvelle observation, fait deux planifications pour chaque objectif potentiel et calcule le coût de chacun des plans. La première planification est forcée de suivre les observations. La seconde planification est forcée de ne pas suivre l'ensemble des observations (il est possible de suivre partiellement les observations). Après les deux planifications effectuées, une valeur d'entropie est calculée en fonction du coût de chacun des deux plans. L'entropie sera élevée si la première planification possède un coût plus faible que la seconde (et respectivement une valeur faible dans le cas inverse). La fonction d'entropie est monotone ce qui signifie que plus la différence est élevée plus la valeur sera élevée (respectivement

faible). Une fois que cette valeur d'entropie est calculée pour chacun des objectifs possibles, toutes les valeurs obtenues sont normalisées afin d'avoir les probabilités de chacun des plans.

Le principal avantage de PR-Plan provient de l'utilisation d'une approche générative. En effet, cela le rend peu sensible au bruit, car une action inutile affecte de la même façon le coût de l'ensemble des plans. Cela le rend aussi résistant à la tromperie, pour les mêmes raisons que le bruit, c'est-à-dire qu'une manœuvre de tromperie risque de l'induire en erreur temporairement, le temps que l'agent effectue à nouveau une action de son plan initial. Le fait de devoir définir uniquement le domaine et les objectifs n'est pas forcement évident, mais est hautement réutilisable. En effet, le découplage du domaine, des objectifs et des capacités de l'agent font qu'une fois les capacités de l'agent modélisées, il est aisé de changer d'environnement. Pas besoin de faire de nouvelles bibliothèques de plans.

PR-Plan possède aussi de grandes faiblesses. Le temps de calcul peut être extrêmement long. En effet, l'algorithme doit faire un grand nombre de planifications sur le domaine, ce qui peut être complexe. De plus, PR-Plan a beaucoup de difficulté à modéliser un agent non rationnel. Si l'on voulait observer un agent non rationnel, il serait nécessaire de faire un nouveau planificateur qui agirait de façon semblable à l'agent. Ensuite, le fait que la probabilité d'un plan ne soit pas calculée automatiquement rend très difficile, voire impossible, de détecter des plans entrelacés.

Chapitre 2

Description du domaine : StarCraft

StarCraft est un jeu de stratégie en temps réel développé en 1998 par *Blizzard Entertainment*. Nous allons faire une description du jeu et voir le lien entre StarCraft et la reconnaissance de plan. Ensuite, nous décrivons un exemple de domaine avec des expérimentations sur des cartes aléatoires. Enfin, nous expliquons la motivation d'observation *in situ* dans le jeu StarCraft.

2.1 Description de StarCraft

StarCraft est un jeu de stratégie en temps réel dont le but est d'anéantir militairement l'adversaire. Une partie se joue de deux à huit joueurs. Les parties peuvent être jouées en équipes ou individuellement.
Les joueurs s'affrontent sur une carte prédéfinie dans laquelle ils vont construire des bases afin de développer une économie qui va leur permettre de faire des recherches technologiques et produire des unités civiles comme militaires afin de prendre le dessus sur l'adversaire. Un élément essentiel du jeu est le fait que les ressources ne sont pas renouvelables et qu'elles sont à des endroits spécifiques connus des joueurs. Cela implique que ces zones possèdent un très fort intérêt stratégique et que l'essentiel du jeu va être une succession d'attaques ou de prises de contrôle de ces zones.

Ensuite, bien que les joueurs connaissent la carte, celle-ci possède un brouillard de guerre qui limite la vision des joueurs aux alentours de leurs unités. On constate que la vision est un élément essentiel du jeu, car elle devient un enjeu important pour savoir ce que fait l'adversaire. En effet, savoir ce que fait l'adversaire permet d'anticiper ses menaces pour mieux les contrer quand elles surviennent. Par exemple si je sais qu'un joueur possède une base contenant uniquement des unités civiles qui participent activement à son économie, cela deviendra une cible de choix pour une attaque rapide visant simplement l'affaiblissement de son économie. Savoir qu'un joueur développe une technologie d'unités invisibles permet de préparer des contre-mesures appropriées avant qu'il ne soit trop tard.
Enfin, le jeu se déroule en temps réel donc les joueurs doivent réagir rapidement et n'ont pas le temps de réfléchir longtemps avant de prendre une décision.

2.2 StarCraft et la reconnaissance de plan

Le domaine comporte un attrait important pour la reconnaissance de plan. L'environnement est hautement dynamique et temps réel. Les joueurs ont systématiquement besoin de surveiller les actions des adversaires afin de connaître leurs plans et leurs stratégies pour pouvoir se protéger ou les contrer.
Il est possible de faire de la reconnaissance de différents types d'intentions dans le jeu. Un premier aspect de reconnaissance se situe dans la stratégie générale du joueur. Par exemple, savoir quels bâtiments il construit, ou quelles technologies il développe.
Un autre aspect de la reconnaissance se retrouve dans le fait que les joueurs sont conscients de la carte et des positions des ressources stratégiques (en quantités limitées). Cela signifie que la prise de contrôle et la défense de ces zones sont majeures pour s'assurer la victoire. Donc la détection et la compréhension des mouvements de troupes ennemies sont primordiales. Il est donc important pour un joueur d'être capable d'attaquer des zones faiblement défendues et de défendre fortement des zones sur le point d'être attaquées. Dans le cadre de ce mémoire, on se concentre principalement sur ce niveau de reconnaissance.
Un troisième aspect de la reconnaissance se retrouve dans les combats directement.

En effet, lorsque deux groupes d'unités engagent un combat ils vont établir des stratégies pour limiter leurs pertes et maximiser les dégâts sur l'ennemi. Reconnaître la stratégie de l'adversaire permet de définir une stratégie qui permettra de savoir sur quel type d'unités focaliser ses actions et lesquels ignorer. Lors d'une partie complète, un joueur doit gérer ces différentes couches du jeu de façon interreliée. En effet, le fait de connaître les technologies de l'adversaire permet de savoir quelles unités sont les plus dangereuses lors d'affrontements. De même, connaître les objectifs de recherche ou les constructions permet de choisir quoi attaquer afin d'empêcher une recherche ou nuire à la production de ressources spécifiques.
Enfin, l'aspect temps réel du jeu implique que chaque joueur prend des décisions simultanément et en continu. Cela signifie que la prise de décision doit être rapide et la capacité d'adaptation des joueurs est cruciale. La reconnaissance de plan doit donc être de qualité et rapide.
On remarque donc que les diverses caractéristiques du domaine semblent convenir parfaitement pour l'utilisation d'algorithme de reconnaissance de plan, car celle-ci influencera directement la qualité des décisions prises par le joueur. De plus, la complexité du domaine rend la tâche de reconnaissance complexe et l'aspect temps réel du jeu oblige une certaine efficacité des algorithmes.

2.3 Cartes aléatoires

Le domaine de StarCraft est complexe et dynamique ce qui rend difficile la comparaison d'algorithmes de reconnaissance de plan. Donc dans un premier temps, il a été décidé d'extraire des caractéristiques importantes du jeu et de faire une première évaluation sur celles-ci.
On se concentre sur la reconnaissance des mouvements de troupes. Donc nous avons analysé plusieurs cartes de StarCraft pour nous rendre compte qu'il est possible de les représenter sous forme d'un graphe non orienté, chaque région de la carte formant un nœud et chaque jonction entre les régions formant un arc. La figure 2.1 représente un exemple de carte de StarCraft avec le graphe de région superposé dessus.

Ensuite, nous avons constaté que dans une partie les unités se déplacent rarement seules, mais plutôt en groupes. Le plus souvent, ces unités vont d'une région à une autre dans un but spécifique, par exemple attaquer une région ennemie, prendre le contrôle d'une région contenant des ressources, *etc.*

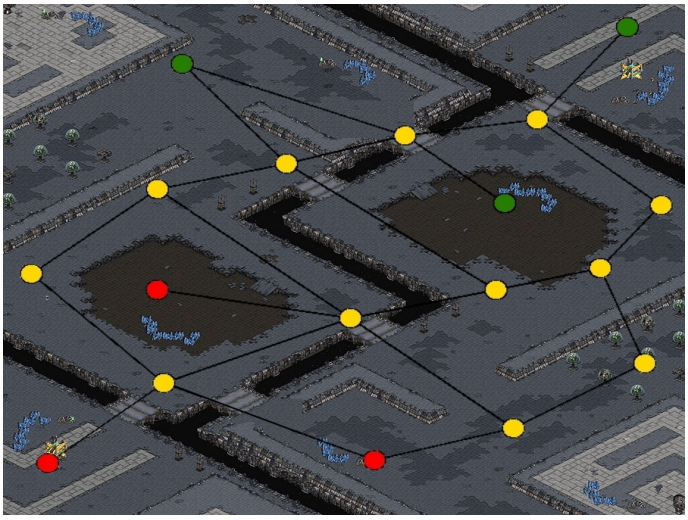

figure 2.1 – Carte de StarCraft *Astral Balance* avec la représentation en graphe non orienté

Ramìrez et Geffner ont, dans leur article [16], défini un domaine de déplacement sur une grille. Le scénario est composé d'un agent qui se déplace sur une grille avec plusieurs destinations potentielles comme sur la figure 2.2. Ce domaine comporte des caractéristiques similaires à notre problème de mouvement de troupes. La différence vient simplement de la modélisation de la carte sous forme de grille au lieu de graphe non orienté.

En effet, sur la carte de StarCraft (figure 2.1) les points verts représentent les bases alliées et les points rouges les bases ennemies. Cela signifie que les unités vont avoir pour plan potentiel de se rendre dans l'une de ces régions. Dans l'exemple de Ramìrez et Geffner (figure 2.2) l'agent peut se rendre à la zone A, B, C, D, E ou F. Ainsi, nous avons adapté le domaine de Ramìrez et Geffner pour nos simulations simples.

Ainsi, notre domaine pour faire des simulations simples est constitué d'un graphe non orienté représentant la carte avec diverses destinations potentielles, un point de

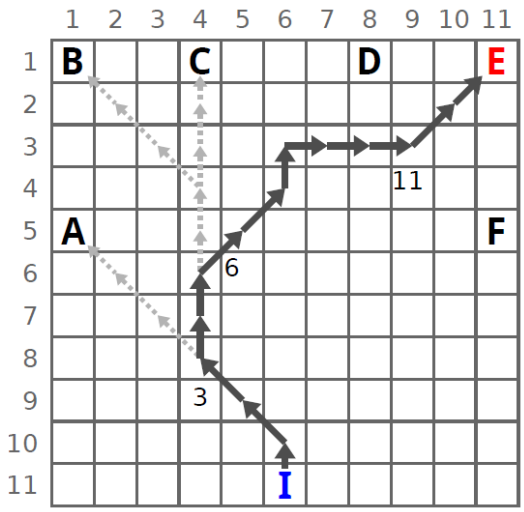

figure 2.2 – Example de domaine de déplacement fait par Ramírez et Geffner pour PR-Plan

départ et une séquence d'observations.

Les cartes de StarCraft étant nombreuses et variées, nous avons décidé pour nos simulations simples de générer aléatoirement des cartes et des objectifs. Puis à partir de cela, nous avons réussi à générer dynamiquement des descriptions de bibliothèques de plan pour YAPPR et d'actions primitives pour PR-Plan.

De cette façon même si le domaine était plus simple que le problème visé nous avons pu avoir un aperçu de la complexité du futur problème. De plus, cela nous a forcé à réussir à générer dynamiquement des domaines pour PR-Plan et des bibliothèques de plans pour YAPPR. Cette étape est importante, car elle nous permet d'intégrer ces algorithmes avec l'agent *in situ*.

Finalement, nous avons comparé les deux algorithmes sur un même problème. Les résultats obtenus sur les cartes aléatoires ont permis de s'assurer que les deux algorithmes reconnaissent le plan exécuté avec un faible taux d'erreur comme nous le constatons plus en détail dans le chapitre 4.

Chapitre 3

Agent de comparaison

La création d'un agent pour un jeu tel que StarCraft est complexe et fait l'objet de travaux de recherche en soi. Il est donc convenu que nous ne ferons pas un agent complet. Ici, nous présentons d'abord les hypothèses sous-jacentes à un agent qui utilise la reconnaissance de plan dans StarCraft. Ensuite, nous précisons les scénarios considérés pour nos expérimentations ainsi que notre agent. Finalement, nous décrivons l'architecture de l'agent mise en place pour effectuer nos comparaisons.

3.1 Reconnaissance de plan et observation *in situ*

Le succès des expérimentations sur les cartes aléatoires nous a donné confiance quant à la possibilité de faire de la reconnaissance de plan sur une partie de StarCraft. StarCraft fut un jeu très populaire de sorte qu'il existe une grande quantité d'enregistrements de parties. Ces enregistrements pourraient constituer un ensemble de problèmes à résoudre qui permettrait la comparaison de nos agents. Cependant, il y a un obstacle majeur à cela : l'étiquetage des données. En effet, dans un domaine théorique il est facile de connaître le plan de l'agent, mais dans une partie réelle l'environnement est dynamique et le joueur change constamment de plan pour s'adapter à la stratégie de son adversaire ; il exécute aussi plusieurs plans simultanément. Cela rend la tâche d'étiquetage d'une partie très complexe

même pour un expert. En fait, la seule façon valable d'étiqueter les données serait de rester à proximité d'un joueur et de prendre en note ce qu'il nous dit. Cependant, cela serait complexe et certainement pas sans erreur. De plus, cela serait excessivement long de créer un ensemble de données de la sorte. Pour cette raison, nous avons décidé de ne pas prendre d'enregistrements de parties.

Ensuite, les expérimentations sur les cartes aléatoires ont confirmé que la reconnaissance de simple déplacement sur une carte est une tâche réussie facilement par les deux algorithmes (*cf.* figure 4.1). La différence entre les cartes aléatoires et les parties de StarCraft proviennent des abandons de plan. En effet, lors des expérimentations sur les cartes aléatoires il n'y a pas d'abandon de plan. Or dans une partie de StarCraft, les plans des joueurs sont contrariés par les actions de leur(s) adversaire(s) et donc les plans sont souvent abandonnés. Le problème est que les algorithmes de reconnaissance de plan choisis ne détectent pas bien les abandons de plans. Il nous est donc nécessaire d'avoir un module qui déterminera les abandons de plans afin de réinitialiser les algorithmes ; de là vient l'intérêt de la génération dynamique de domaines. C'est-à-dire être capable de générer un domaine de reconnaissance de plan pour les algorithmes utilisés de façon dynamique. Dans le cas de YAPPR on reconstruit la bibliothèque de plan, alors que pour PR-Plan on construit un domaine PDDL avec l'ensemble de positions et des objectifs potentiels.

De plus, dans le jeu les joueurs possèdent plusieurs groupes d'unités avec des intentions différentes pour chacun d'entre eux. Pour gérer cette situation nous avons décidé de détecter les groupes d'unités et d'avoir une instance d'algorithme de reconnaissance de plan pour chacun des groupes adversaires. De cette façon, les algorithmes n'ont pas besoin de détecter la multiplicité des plans ni de désentrelacer les actions.

Nous faisons alors les hypothèses suivantes :

Hypothèse 3.1.1 *Chaque groupe d'unités exécute un seul et unique plan à un moment donné.*

Hypothèse 3.1.2 *La reconnaissance de plan a un impact important sur le déroulement du jeu et les choix des joueurs.*

Donc une façon de comparer les algorithmes de reconnaissance de plan est de faire s'affronter des agents dans des parties de StarCraft avec pour seule différence leurs modules de reconnaissance de plan. Cette façon de procéder rappelle les compétitions d'intelligence artificielle organisées par AIIDE dans lesquelles différentes intelligences artificielles s'affrontent dans des parties de StarCraft. L'agent qui gagne le plus souvent est considéré comme le meilleur.
Alors, nous avons décidé de faire une observation *in situ*, c'est-à-dire de faire un agent qui joue à StarCraft avec un module de reconnaissance de plan. Cet agent devra agir en fonction des plans qu'il reconnaît. Ainsi un agent qui aura une reconnaissance de plan plus efficace devrait prendre de meilleures décisions que son homologue moins efficace et ainsi le vaincre. Ceci nous mène à notre troisième hypothèse :

Hypothèse 3.1.3 *Une meilleure reconnaissance des plans de l'adversaire rend le joueur meilleur.*

Cette hypothèse dépend bien sûr de la façon dont l'IA (le module de prise de décision) des agents est construit et de la façon dont il exploite la reconnaissance de plan. Nous n'avons pas de caractérisation théorique sur les architectures d'IA satisfaisant cette hypothèse. Nous avons plutôt procédé de façon empirique, en construisant une IA particulière et en validant par expérimentation si l'hypothèse semble satisfaite.
Pour valider notre hypothèse, il faudrait comparer deux agents, l'un possédant une reconnaissance de plan et l'autre non. Cependant, une telle comparaison est complexe, puisque même deux agents sans reconnaissance de plan ne sont pas de même qualité et faire un agent intelligent complet pour StarCraft est complexe. Donc afin de montrer que notre hypothèse semble valide sans toutefois pouvoir la prouver, nous avons créé deux versions différentes de la même IA. Le premier utilisant une reconnaissance de plan parfaite et le second ayant une reconnaissance de plan purement réactive. La reconnaissance de plan réactive signifie que l'algorithme ne se rend compte du plan de l'adversaire seulement lorsque celui-ci est terminé. Par exemple, il reconnaîtra que le plan de l'adversaire est d'attaquer une base uniquement lorsque celle-ci est sous le feu ennemi. Ainsi si notre hypothèse est

véridique, notre agent avec reconnaissance parfaite devrait gagner de façon écrasante face à l'agent avec reconnaissance réactive. Aussi, notre agent avec reconnaissance parfaite devrait être égal ou supérieur à n'importe quel autre algorithme puisqu'il est impossible d'avoir une meilleure reconnaissance. Après plusieurs simulations, on a pu observer que les résultats étaient conformes à nos attentes. Nous avons donc pu faire jouer nos agents les uns contre les autres afin d'observer le taux de victoire de chacun et ainsi déterminer la qualité de chacun d'entre eux.

3.2 La création des scénarios

StarCraft est un jeu complexe avec plusieurs possibilités pour la reconnaissance de plan. Pour nos scénarios les agents vont s'affronter avec pour but de détruire le plus de bâtiments ennemis et d'en perdre le moins possible. La détermination du plan de l'adversaire permettra de déterminer les cibles de l'adversaire avant son arrivée et ainsi de défendre correctement les bases pour limiter les pertes de bâtiments. Ensuite, la détermination des plans défensifs de l'adversaire va permettre d'attaquer des zones faiblement défendues et fuir avant l'arrivée des défenseurs. Pour simplifier les scénarios, nous avons décidé d'exclure volontairement toute microgestion, c'est-à-dire la création de bâtiments, la gestion des ressources, *etc.* Le seul comportement supporté par l'agent est le recrutement d'unités afin de renforcer un groupe d'unités. Ce choix provient du fait que ce sous-domaine est un sous-ensemble du jeu complet. En effet, une intelligence artificielle complète pourrait intégrer cet agent comme un simple module qui gère les unités de combat afin de savoir où attaquer et où défendre.

3.2.1 Choix des paramètres

Une fois le scénario établi dans les grandes lignes, nous proposons d'analyser les variables qui entrent en compte telles que la vitesse d'exécution, le nombre de groupes, la composition des groupes, la position des groupes, le nombre de bases, la position des bases, les bâtiments dans chacune des bases et la carte.

Pour déterminer les valeurs de ces variables, nous avons décidé de le faire de façon empirique. Étant donné que l'exécution d'un scénario peut-être longue, il n'est pas envisageable de déterminer autrement ces attributs.

Pour générer des échantillons de scénarios de comparaison, nous avons fait plusieurs simulations en modifiant les paramètres de sorte à modifier les comportements des agents, tout en comparant les algorithmes de reconnaissance de plan sur chaque comportement. Nous avons décidé de choisir une vitesse d'exécution du jeu similaire à celle pour les humains, de sorte que le délai pour les algorithmes de reconnaissance de plan était celui attendu dans une situation de jeu réaliste. Une vitesse trop élevée telle que pratiquée dans les compétitions d'intelligence artificielle sur StarCraft laisse très peu de temps aux algorithmes pour détecter un plan et pourrait conduire à des conclusions différentes quant à l'efficacité des différents algorithmes.

La sélection de la carte est une étape importante. La carte doit contenir suffisamment de régions et de connexions entre les régions pour être intéressante. En effet, une carte avec un seul chemin possible sous forme de couloir ne représente aucun intérêt pour la reconnaissance de plan.

Le choix du nombre de base par joueur est déterminant. Une seule base à défendre ne représente pas d'intérêt, car on connaît aisément la cible de l'adversaire. Deux bases rendent la reconnaissance utile, mais relativement facile, elle se résume à un choix binaire. Trois bases ajoutent plus de complexité que deux sans pour autant nécessiter une carte excessivement grande. Nous avons pu observer que le fait d'avoir plus de trois bases ne semble pas changer beaucoup au jeu si ce n'est que les cartes deviennent grandes, donc les scénarios sont plus longs, mais les résultats semblent identiques. C'est pour cette raison que nous avons choisi d'avoir trois bases pour chaque joueur.

La composition des groupes d'unités est importante, car chaque unité possède une grande quantité d'attributs variables tels que la vitesse de déplacement, les dégâts, la portée, les points de vie, la taille, *etc.* Nous avons donc choisi une unité commune et plutôt faible dans le jeu : les *Marines*. Ce choix s'explique par plusieurs raisons. En effet, cette unité est souvent utilisée dans de vraies parties. Aussi, sa vitesse de déplacement plutôt lente par rapport aux autres unités du jeu fait qu'elle prend du temps pour se rendre dans une région lointaine. Ainsi une mauvaise reconnaissance

de plan provoque un délai avant de pouvoir revenir défendre une base alliée.

La sélection du nombre de groupes d'unités par joueur est un facteur important pour deux raisons. Le nombre de groupes d'unités détermine le nombre d'algorithmes de reconnaissance qui fonctionnent en parallèle. Cela signifie qu'un nombre de groupes trop important risque de poser des problèmes de performance non pas à cause des algorithmes eux-mêmes, mais du nombre d'instances qui s'exécuteront en parallèle. Or on ne cherche pas à évaluer la capacité des machines de tests, mais des algorithmes. Ensuite, la quantité de groupes détermine la quantité de plans qui peuvent être menés en parallèle. Cela signifie qu'avoir un seul groupe n'apporte que peu d'intérêt, car dès lors que l'agent sera attaqué il ne pourra rien faire à part se défendre et les scénarios seront très courts. L'issue des combats entre groupes étant aléatoire, le fait d'avoir plus de groupes réduit l'impact de l'aléatoire des combats. Après plusieurs tests, il a été décidé de prendre trois groupes.

Le nombre d'unités par groupe a aussi un impact, parce que des groupes trop petits rendent la prise de décision trop simple pour nécessiter la planification et donc la reconnaissance de plan : le premier qui tire est celui qui gagne. *A contrario*, des groupes trop gros rendent la gestion de groupes d'unités dans un combat plus complexe ; or notre évaluation ne se situe pas en matière du combat. Pour cette raison, nous avons choisi des groupes de quatre unités.

3.2.2 Les scénarios choisis

Considérant les éléments précédents, nous avons créé deux scénarios pour nos comparaisons. Le premier scénario se joue sur une carte officielle de StarCraft, fournie avec le jeu. Chacun des deux joueurs possède trois bases et trois groupes de quatre unités qui sont rassemblés dans l'une des bases. Le second scénario se joue sur une carte créée le plus symétriquement possible, dans l'optique de faire une carte équitable. Chacun des deux joueurs possède également trois bases et trois groupes de quatre unités.

3.2.3 Les requis de l'agent

Une fois les scénarios bien définis, nous avons extrait les requis pour notre agent. Les éléments pertinents pour notre agent sont les mouvements d'unités et le combat. Pour ce faire, l'agent supporte le déplacement de groupes d'unités, le combat d'un groupe d'unités contre un autre groupe d'unités ou contre un bâtiment, le renforcement d'un groupe d'unités par le recrutement de nouvelles unités.

3.3 Architecture de l'agent

Une fois les scénarios et les requis de l'agent déterminés, nous avons réfléchi à l'architecture pour l'agent. L'architecture choisie se décompose en trois modules principaux à responsabilité claire et distincte. Le module d'observation, le module de reconnaissance de plan et le module de prise de décision. Cette architecture est importante, car elle permet l'amélioration des divers modules de l'agent de façon indépendante, et considérant que la création d'un agent est complexe cela permet un meilleur potentiel d'évolution pour l'agent. Cela fonctionne plutôt bien dans la mesure où notre agent comporte quatre systèmes de reconnaissance de plan sans avoir de code spécifique pour chacun des algorithmes dans notre agent.

3.3.1 Module d'observation

Le module d'observation a pour vocation de fournir de l'information sur l'état du jeu, c'est une couche d'abstraction par-dessus le jeu. Cela nous permet d'extraire différentes informations de plus haut niveau. Le module d'observation va reconnaître les groupes d'unités qui se déplacent ensemble. Il sépare la carte en différentes régions de plus haut niveau. Il détermine les régions qui contiennent des unités ennemies, les bases ennemies, les bases alliées, les lieux d'intérêt. Ce module est très important, car il permet de faire des abstractions du jeu qui autorisent la détection d'actions de plus haut niveau plus pertinentes pour les algorithmes de reconnaissance de plan. Cela permet de réduire le nombre d'actions observées et ainsi de rendre le travail des algorithmes de reconnaissance de plan envisageable. Ce module assiste aussi le module de prise de décision, car il fournit une analyse des

informations et permet de prendre de meilleures décisions. Il calcule par exemple l'estimation du temps requis par un groupe pour se déplacer d'une région vers une autre, chose particulièrement importante si l'on veut savoir quand fuir.

3.3.2 Module de reconnaissance de plan

Le module de reconnaissance de plan fait l'intégration entre les algorithmes de reconnaissance de plan et l'agent de StarCraft. Il définit une interface de communication avec les algorithmes ce qui permet de s'assurer que les algorithmes sont comparés avec la même information.
Le module fonctionne comme suit. En début de partie, il envoie de l'information générale sur le scénario. Le nombre de bases, leur position, les liens entre les régions, la position initiale des unités, *etc.* Cela permet à chacun des algorithmes d'initialiser les éléments dont il a besoin. Ensuite, à chaque cadre de jeu le sous-module de gestion des observations est notifié, celui-ci regarde si des actions se sont produites (on parle ici d'actions de haut niveau), si oui il les envoie aux algorithmes, sinon il ne fait rien.

Gestion des algorithmes

Le sous-module de gestion des algorithmes crée une instance de l'algorithme choisi pour chaque groupe d'unités adverses observé. La gestion des instances se fait dans ce sous-module qui filtre les informations pour chacune des instances. En effet, il détermine à quelle instance chaque observation doit se rendre. Mais il s'occupe aussi de la communication avec les algorithmes. Cela permet de découpler le reste de l'agent de la communication avec les algorithmes de reconnaissance de plan.

Gestion des observations

Le sous-module de gestion des observations utilise le module d'observation afin d'extraire des observations de plus haut niveau (e.g. *unit-move-from-a-to-b*) qui sont envoyées aux algorithmes de reconnaissance de plan, mais aussi au sous-module de détection des abandons de plan.

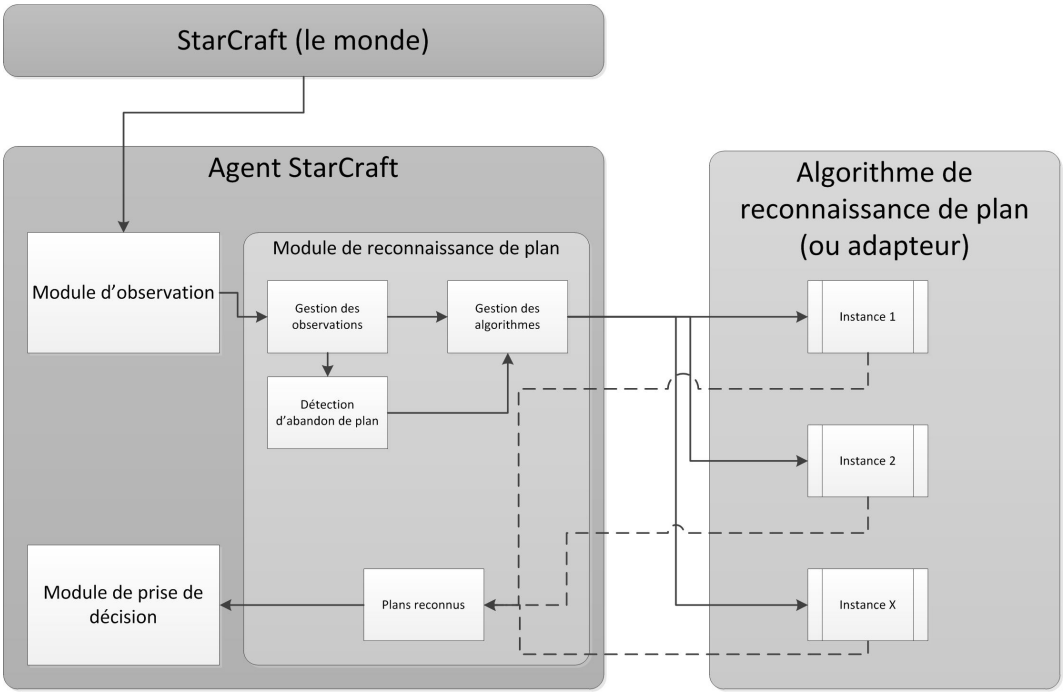

figure 3.1 – Module de reconnaissance de plan

Détection d'abandon de plan

Les algorithmes de reconnaissance de plan ne supporte pas ou très mal l'abandon de plan. Il est décidé de faire un module qui s'occupe de la détection d'abandon de plan. Cela permet d'exclure un facteur de notre comparaison.

Le sous-module de détection d'abandon de plan reçoit des observations et détermine si l'agent a abandonné son plan ou non. Comme on cherche à comparer les algorithmes de reconnaissance de plan on a décidé de faire une implémentation parfaite de détection d'abandon de plan. L'implémentation courante est une détection parfaite des abandons de plan, car chaque agent notifie l'autre agent (son adversaire) de son nouveau plan. Cette implémentation nous permet ainsi d'avoir la connaissance, en tout temps, du plan de l'adversaire qui est nécessaire pour la création d'un agent à reconnaissance parfaite. Cette connaissance du plan de l'adversaire est utilisée exclusivement par l'agent parfait. Les autres implémentations ont simplement connaissance qu'il y a ou non un abandon de plan.

Plan reconnu

Le sous-module de plans reconnus masque l'aspect asynchrone de la reconnaissance de plan. En effet, les algorithmes de reconnaissance de plan sont rarement instantanés. Pour cette raison, le sous-module garde toujours les derniers résultats obtenus par les algorithmes de reconnaissance de plan, et met à jour les plans reconnus aussitôt qu'une instance lui retourne un résultat.

3.3.3 Adaptateur pour les algorithmes

Nous avons dû créer des adaptateurs pour chaque algorithme afin de pouvoir communiquer de façon identique avec notre agent. Ces adaptateurs ont pour fonction d'initialiser les algorithmes correctement, c'est-à-dire de générer leurs domaines (bibliothèque de plans ou domaine PDDL). Ils doivent aussi soumettre de nouvelles observations et analyser les résultats générés par les algorithmes.
Nous avons pu récupérer le travail effectué pour les cartes aléatoires, c'est-à-dire que l'on génère les bibliothèques de plans et les domaines PDDL dynamiquement lors du début du scénario, puis lors de chaque changement de plan du groupe observé.

3.3.4 Module de prise de décision

Le but de ce module de prise de décision est de déterminer les objectifs pour chacun des groupes d'unités en fonction des plans de l'adversaire et de l'état du jeu. Puis, une fois les objectifs déterminés, il choisit un plan et l'exécute. Il interagit directement avec le jeu. Ce module est important pour la qualité de l'agent, car un mauvais système de prise de décision rendra les algorithmes de reconnaissance de plan inutiles.
Il est convenu que le système de prise de décision sera simple et sujet à amélioration dans des travaux futurs dans la mesure où notre objectif ici est de poser les bases de la comparaison d'algorithmes par l'observation *in situ*. Nous avons décidé de considérer uniquement le plan de plus haute probabilité comme le plan reconnu, et ce, quelle que soit la probabilité que ce plan soit le bon (par exemple si les probabilités sont : A 27%, B 23%, C 25%, D 25%, on considère que le plan reconnu

est le plan A). Cela signifie qu'on ne tire pas avantage des algorithmes tels que YAPPR qui donnent des probabilités plus précises pour chacun des plans. Cela est encore plus dommage pour des algorithmes tels que DOPLAR qui permettent de borner les probabilités.

Pour cette raison, nous avons utilisé une simple règle de décision sachant que des travaux futurs pourraient améliorer ce processus de prise de décision à l'aide de l'architecture modulaire choisie.

La détermination des objectifs pour chacun de nos groupes se fait en accord avec la règle de décision suivante :

> *Si au moins un adversaire menace une base (le plan d'un groupe d'unité adverse est d'attaquer une base alliée), alors on choisit le groupe allié le plus proche de cette base pour la défendre. Le groupe choisi va tenter de défendre la base tant que celle-ci n'est pas détruite. Si la base est détruite alors l'unité va changer de plan. Les autres groupes vont attaquer une base ennemie non défendue et dont le défenseur est plus loin que le temps nécessaire à fuir la base ciblée.*

Une fois les objectifs sélectionnés, il est nécessaire de faire un plan pour atteindre nos objectifs. Pour ce faire, nous avons fait des ensembles d'actions autonomes composés de sous-actions. Par exemple, l'action *attaquer une base* se décompose en : *se déplacer de façon sécuritaire de notre position vers la région ennemie*, puis *attaquer la région ennemie*. L'action *se déplacer de façon sécuritaire de notre position vers une autre région* est décomposée en un ensemble d'actions qui consiste à *se déplacer d'une région vers la suivante*, jusqu'à atteindre les actions de bas niveau qui spécifient à chaque unité quoi faire (par exemple, *se déplacer à la position X* ou *attaquer l'unité Y*). Cette architecture de commandement sous forme de HTN permet à notre agent de prendre des décisions de haut niveau dont les comportements spécifiques sont implémentés de façon indépendante. Ainsi, bien que ce ne soit pas le cœur de ce travail que de concevoir un agent avec des comportements optimaux, ceux-ci pourraient être améliorés sans impact vis-à-vis de l'architecture existante.

Pour conclure, après plusieurs tests empiriques nous avons remarqué qu'en dépit de sa simplicité nous avions des résultats intéressants. Cela nous a confortés dans la

décision de ne pas mettre plus d'effort afin d'améliorer ce module dans un premier temps.
Une fois la réalisation de cet agent complétée et testée, nous avons mené à bien des expérimentations ou notre agent s'affronte lui-même avec les divers algorithmes de reconnaissance de plan.

3.4 L'intégration des algorithmes

Durant la création de l'agent, nous avons donc déterminé un ensemble de fonctionnalités standards que les algorithmes de reconnaissance de plan devaient implémenter.
Nous avons récupéré des implémentations existantes des algorithmes. Chacun de ces algorithmes s'exécute dans des langages différents et fonctionne de façon différente. Nous avons donc créé des applications Java afin d'encapsuler ces applications. Notre encapsulation permet d'offrir une interface standard de communication avec les algorithmes de reconnaissance de plan. Faire cela n'était pas évident à cause de la grande différence des interfaces de chacun de ces algorithmes. En effet, l'implémentation de YAPPR ainsi que la définition des domaines sont en LISP. De plus, dans le cas de YAPPR il est important de conserver l'état de l'algorithme, car l'ajout d'une observation n'est pas équivalent au calcul de l'ensemble des observations. D'un autre côté, PR-Plan est un ensemble de programmes en Python, C++ et PDDL qui fonctionne exclusivement sur Linux. PR-Plan ne nécessite pas de conserver un état, car il fait des planifications sur les domaines PDDL et modifie les domaines selon les observations. Ainsi PR-Plan prend un temps de calcul indépendant du nombre d'observations.

Chapitre 4

Expérimentation

Dans ce chapitre, nous décrivons les expérimentations faites. En premier lieu, nous parlons des métriques utilisées dans le cadre de nos expérimentations. Ensuite, nous allons décrire les expérimentations effectuées sur les cartes générées aléatoirement. Et finalement, nous analysons les résultats obtenus lors de nos expérimentations *in situ.* Lors des expérimentations nous cherchons à montrer que notre système de comparaison *in situ* est valide. Puis de voir quel algorithme de reconnaissance de plan se comporte le mieux.

4.1 Métriques

4.1.1 Matrice de confusion

Pour comparer les résultats des divers algorithmes sur les scénarios de cartes générés aléatoirement nous avons utilisé une méthodologie similaire à celle de Ramírez et Geffner. Nous avons converti les résultats en probabilités et considéré uniquement la plus haute probabilité comme prédiction. De cette façon, il est possible de faire une classification binaire et créer une matrice de confusion.

Tout d'abord, nous définissons certains termes liés aux matrices de confusion :

- les Vrais Positifs (VP) comme le nombre de fois où le plan prédit est le plan exécuté, c'est-à-dire le nombre de prédictions correctes.

		Plan prédit	
		Vrai	Faux
Plan réel	Vrai	VP	FN
	Faux	FP	VN

tableau 4.1 – Example d'une matrice de confusion

- les Faux Négatif (FN) comme le nombre de fois que le plan exécuté n'est pas prédit comme le plan le plus probable, c'est-à-dire que la prédiction est erronée.
- les Faux Positif (FP) comme le nombre de fois où le plan prédit n'est pas le plan exécuté, c'est-à-dire que la prédiction est erronée.
- les Vrai Négatif (VN) comme le nombre de fois où le plan qui n'est pas prédit n'est pas le plan exécuté, c'est-à-dire le nombre de prédictions correctes fausses.

Les matrices de confusion sont souvent utilisées pour la classification. Nous allons utiliser les métriques les plus fréquemment utilisées, définies à partir d'une matrice de confusion (voir Tableau 4.2 pour les formules) :

— La précision P mesure la qualité des prédictions, c'est-à-dire la proportion de prédictions vraies correctement prédites par rapport à l'ensemble des plans prédits comme vrais.
— Le rappel R mesure la proportion de prédictions vraies correctement prédites par rapport à l'ensemble des plans vraiment exécutés.
— L'exactitude E mesure la proportion de prédictions correctes (c'est-à-dire les VP et les VN) par rapport à l'ensemble des prédictions.
— Le score F1 mesure la moyenne harmonique entre la précision et le rappel.

Chacune de ces mesures est bornée entre 0 qui représente un échec total et 1 qui représente un score parfait. Donc plus un algorithme obtient une valeur qui tend vers 1 dans l'ensemble de ces métriques plus il est considéré comme performant.

Le score F1 est souvent critiqué, car il ne prend pas en compte le nombre de vrais négatifs. Nous avons donc décidé de consolider nos métriques par un dernier indice, le coefficient Kappa.

Métrique	Symbole	Équation
Précision	P	$VP/(VP+FP)$
Rappel	R	$VP/(VP+FN)$
Spécificité	S	$VN/(VN+FP)$
Exactitude	E	$(VP+VN)/(VP+FN+FP+VN)$
Score F1	F_1	$2\times(P\times R)/(P+R)$

tableau 4.2 – Métriques calculées à partir de la matrice de confusion

κ	Interprétation
< 0	Désaccord
0.0 — 0.20	Accord très faible
0.21 — 0.40	Accord faible
0.41 — 0.60	Accord modéré
0.61 — 0.80	Accord fort
0.81 — 1.00	Accord presque parfait

tableau 4.3 – Table d'interprétation pour le coefficient κ

Le coefficient Kappa

Le coefficient Kappa mesure l'accord entre l'algorithme et la réalité. Dans notre cas, cela mesure à quel point l'algorithme modélise correctement la réalité par rapport aux chances qu'il réussisse par hasard.

Le coefficient Kappa se calcule comme suit :

$$\kappa = \frac{(Pr_a - Pr_e)}{(1 - Pr_e)},$$

où Pr_a est l'accord relatif de l'algorithme et Pr_e la probabilité d'un accord aléatoire.

Dans notre situation l'accord relatif de l'algorithme correspond à l'exactitude, c'est-à-dire la proportion de plans correctement étiquetés sur le nombre total de prédictions. Pour la reconnaissance de plan, la probabilité d'un accord aléatoire est de 1 divisée par le nombre de buts potentiel de l'agent.

Le tableau 4.3 établi par Landis et Koch [12] donne une d'interprétation des valeurs du coefficient Kappa (κ).

4.1.2 Intervalle de confiance

Nous avons utilisé le concept d'intervalle de confiance pour définir la marge d'erreur du taux de victoire observé lors de nos simulations. Dans nos figures nous avons utilisé un intervalle de confiance à 95%, c'est-à-dire que nous faisons un encadrement du taux de victoire qui est valide 95 fois sur 100 en moyenne.
On calcule l'intervalle de confiance 95% comme suit :

$$1,96 \times \sqrt{\frac{V \times (1-V)}{T}}$$

Avec $V = \frac{v}{T}$ le taux de victoire moyen, v le nombre total de victoires et T le nombre total de simulations.
L'intervalle de confiance est pertinent dans la mesure où il permet de consolider notre taux de victoire. En effet, la durée des parties empêche de faire une quantité de parties qui enlèveraient tout doute sur les résultats obtenus.

4.2 Cartes aléatoires

Notre première expérimentation consiste à générer un ensemble de cartes aléatoires avec plusieurs objectifs potentiels. Nous allons décrire comment nous avons généré nos cartes avant de voir les résultats obtenus.
La motivation de faire des cartes aléatoires est d'avoir des scénarios semblables à ceux d'une partie de StarCraft. La génération de nombreux scénarios correspond à la contrainte de changement de situations que l'on retrouve dans StarCraft. C'est une première étape avant les expérimentations *in situ*.

4.2.1 Méthodologie

Nous avons généré les cartes en utilisant le modèle de Erdős et Rényi [6]. Nos simulations nécessitant des cartes entièrement connectées, nous avons connecté les nœuds ou ensembles de nœuds qui n'étaient pas connectés ensemble en créant des liens aléatoires.

Nous avons généré 50 cartes, chacune contenant 50 régions. Les régions possèdent un facteur de branchement moyen de 3.5. Pour chacune des cartes générées, nous avons déterminé un point de départ et cinq destinations possibles.
Pour chaque carte nous avons généré l'ensemble des scénarios possibles pour nous rendre du point de départ vers chacune des destinations en suivant un chemin optimal. Cela représente 392 scénarios répartis sur les 50 cartes dont la longueur moyenne des plans est de 5.5 actions.
Une fois les scénarios générés, nous avons bâti dynamiquement les domaines pour chacun des algorithmes. C'est-à-dire dans le cas de YAPPR on a généré la bibliothèque de plan et pour PR-Plan le domaine PDDL et les objectifs possibles. Puis, nous avons exécuté les algorithmes sur l'ensemble des scénarios.

4.2.2 Résultats

Les scénarios étant de longueurs variables, nous avons utilisé le taux de complétion du scénario comme point de référence pour le calcul des différentes métriques. Nous avons analysé les métriques à cinq moments de chaque scénario, soit 10, 30, 50, 70 et 100 pour cent de complétion. La complétion des scénarios est calculée en faisant le ratio du nombre d'actions courantes avec le nombre total d'actions du scénario exécuté. En plus des matrices de confusion, nous avons calculé le temps de calcul moyen de chaque algorithme après chaque observation pour chaque scénario. Les résultats détaillés des scénarios sont présentés dans l'annexe A.
La figure 4.1 représente la moyenne des principales métriques selon la complétion des scénarios.

4.2.3 Analyse des résultats

La figure 4.1, montre bien que sur ces scénarios les résultats sont sensiblement les mêmes pour les deux algorithmes. Elle permet toutefois de montrer les différences entre les deux approches. En effet, l'exactitude, la précision et le coefficient Kappa restent similaires entre les deux algorithmes à mesure que le scénario évolue.
Cependant, on remarque que le taux de faux positif est significativement supérieur avec Plan-PR (voir annexe A), par contre le taux de faux négatif est

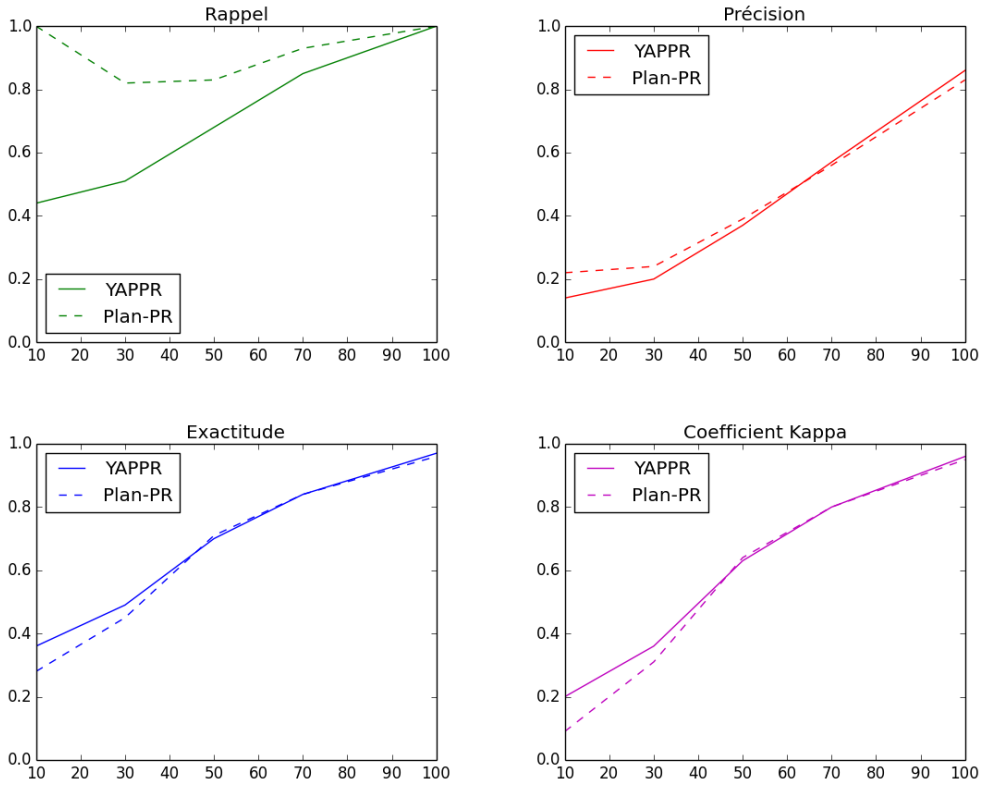

figure 4.1 – Comparaison de YAPPR et Plan-PR sur des cartes aléatoires

significativement supérieur avec YAPPR, on le voit bien avec la moyenne du rappel qui est élevée pour Plan-PR alors qu'elle est plutôt faible pour YAPPR. Quand le scénario est avancé, plus de 70% de complétion, les deux ont sensiblement les mêmes valeurs.

En effet, YAPPR cherche des probabilités exactes pour chacune des hypothèses. Cela fait que plusieurs hypothèses ont rarement la même probabilité. Les probabilités sont plutôt volatiles. Cela fait que la prédiction de plus haute probabilité est souvent erronée, particulièrement en début de scénario (faible taux de vrai positif). Cependant, il réussit plutôt bien à retirer rapidement les plans les plus improbables (taux plus faible de faux positif).

Ces résultats montrent que pour PR-Plan la situation est différente. À l'inverse de YAPPR, PR-Plan garde l'ensemble des hypothèses avec une probabilité équivalente

tant et aussi longtemps qu'il n'aura pas de façon de rejeter un plan improbable. Cela explique le faible taux de faux négatif tout au long de la simulation. C'est aussi pour cette raison que le taux de vrai négatif au début du scénario est faible.
Ces expérimentations nous apprennent un autre élément important : le temps de calcul pour déterminer les probabilités de chacun des plans est fortement plus élevé avec Plan-PR qu'avec YAPPR. En effet, YAPPR prend en moyenne 3.36 millisecondes par observation alors que PR-Plan prend environ 71964.95 millisecondes. Dans le cadre de ces scénarios, la différence de temps de calcul n'a pas d'effet. Mais nous allons voir si cette différence de temps possède un réel impact dans des observations *in situ*, où la vitesse de découverte des plans de l'adversaire peut avoir un impact majeur.

4.3 Agent *in situ*

Les expérimentations *in situ* se sont faites en deux étapes. Dans un premier temps, nous avons exécuté plusieurs simulations afin de voir le comportement de l'agent. Nous avons fait quelques améliorations par exemple la capacité de renforcer un groupe d'unité existante. Nous avons aussi analysé les impacts des différentes variables de simulation. Nous avons fait varié le nombre d'unités par groupe, le nombre de groupes, le type d'unité *etc.* Une fois satisfaits des scénarios créés et du comportement de l'agent, nous avons exécuté un grand nombre de simulations sur ces scénarios.

4.3.1 Méthodologie

Notre environnement de test est plutôt complexe. En effet, StarCraft avec l'API BWAPI ne fonctionne que sur le système d'exploitation Windows. De plus, une seule instance de StarCraft peut fonctionner sur une machine. Ensuite, l'implémentation de Plan-PR ne fonctionne que sur le système d'exploitation Linux. Nous avons donc utilisé deux machines physiques avec Windows 7 et une machine virtuelle avec Ubuntu.
Les scénarios comportent deux joueurs, chacun d'eux est joué par un agent qui

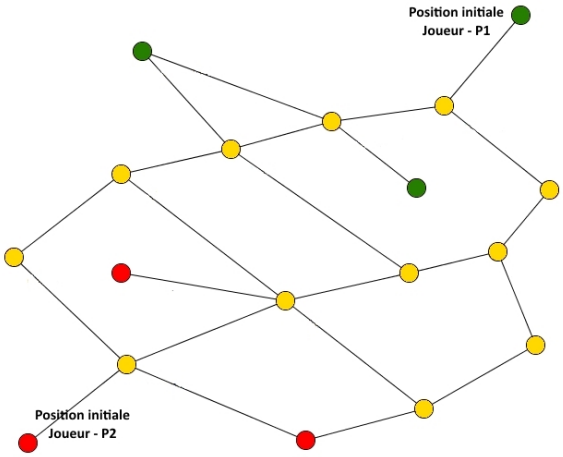

figure 4.2 – Carte *Astral Balance*

utilise un algorithme de reconnaissance de plan. Chacun des joueurs va tenter de détruire l'ensemble des bases ennemies. La partie se termine lorsque l'un des joueurs a perdu l'ensemble de ses unités ou l'ensemble de ses bases.
Lors des simulations nous avons gardé une trace des informations suivantes : le nom des agents, le vainqueur, le nombre de bâtiments restants, les points de vie restants des bâtiments et le nombre d'unités restantes. La condition de victoire de nos scénarios est simple, le vainqueur est l'agent qui a vaincu l'ensemble des unités de l'adversaire ou l'ensemble des bâtiments de l'adversaire.

4.3.2 Scénario 1 : *Astral Balance*

La carte utilisée pour le premier scénario est une carte fournie avec le jeu StarCraft sous le nom de *Astral Balance* que nous avons légèrement modifiée. La figure 4.2 représente la carte telle que perçue par notre agent. Les points verts représentent les bases du joueur $P1$ et les points rouges les bases du joueur $P2$. Dans ce scénario, chaque joueur possède trois groupes de quatre unités qui sont tous dans la base en haut à droite (respectivement en bas à gauche).
Nous avons exécuté un total de 2060 parties, toutes paires d'agents confondues, sur ce scénario, soit environ sept jours de simulation continue. La figure 4.3 présente le taux de victoire obtenu par chacun des agents $P1$ lorsqu'ils affrontent le joueur $P2$

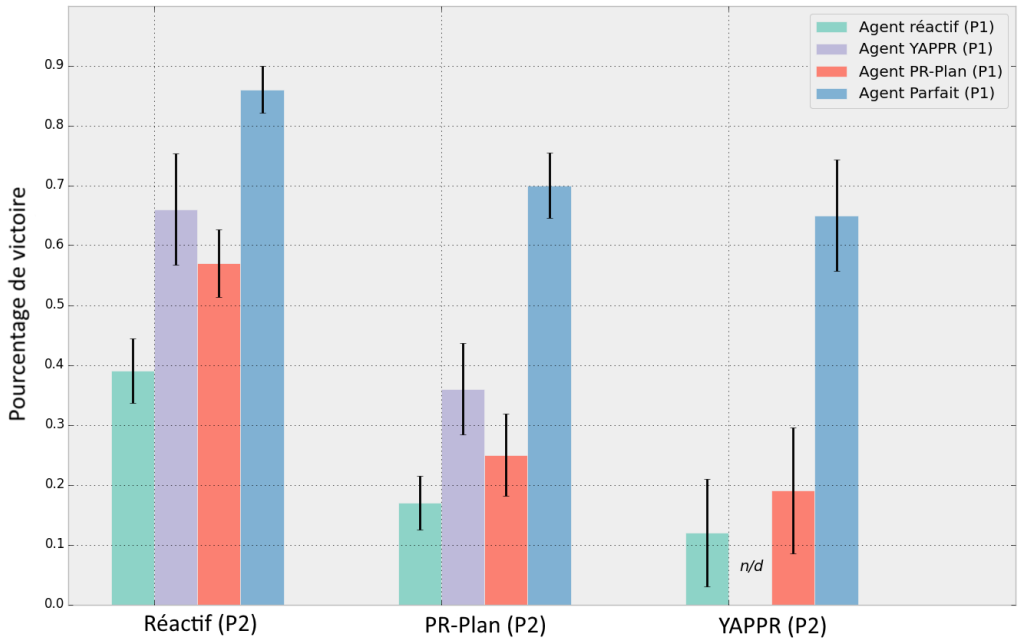

figure 4.3 – Taux de victoire sur la carte *Astral Balance*

utilisant l'algorithme spécifié en abscisse.

Ainsi on voit sur la figure 4.2 le joueur $P2$ avec le module de reconnaissance de plan réactif affronter le joueur $P1$ avec chacun des algorithmes de reconnaissance de plan. Le joueur $P1$ possède les bases vertes sur la carte figure 4.2 et le joueur $P2$ possède les bases rouges.

Tout d'abord, on constate que la carte n'est pas équitable. En effet, lorsque deux agents utilisant le même algorithme de reconnaissance de plan s'affrontent l'agent $P2$ semble avantagé. L'agent du joueur $P1$ avec reconnaissance de plan réactive gagne environ 40% des parties contre l'agent du joueur $P2$ avec le même algorithme de reconnaissance de plan. De même, l'agent du joueur $P1$ avec la reconnaissance de plan PR-Plan gagne environ 25% des parties contre l'agent du joueur $P2$ avec le même algorithme de reconnaissance de plan. L'avantage de la carte n'est pas évident pour un regard non averti, mais un certain nombre de facteurs peuvent avantager un joueur plutôt qu'un autre. Par exemple, la distance entre chacune des bases, la difficulté des chemins, *etc.*

Les résultats montrent que globalement, quel que soit l'adversaire, l'agent avec

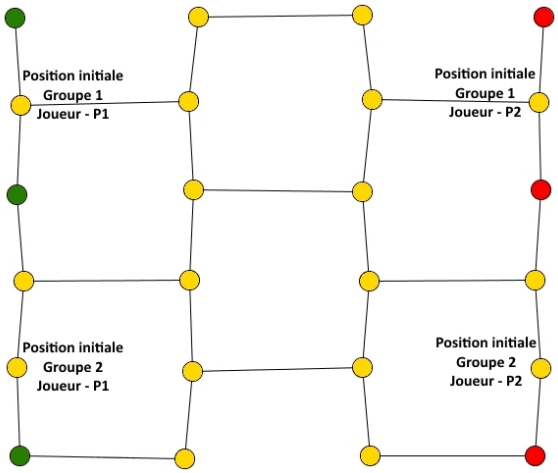

figure 4.4 – Carte *Symmetric Small*

reconnaissance parfaite domine les parties avec un taux de victoire qui dépasse les 65%. À l'inverse, l'agent réactif n'atteint pas les 20% face aux autres algorithmes.

Ce résultat est cohérent avec notre hypothèse selon laquelle un meilleur algorithme de reconnaissance de plan rend notre agent meilleur. Ce qui se traduit par un meilleur taux de victoire.

On remarque aussi que les agents avec PR-Plan et YAPPR dépassent nettement les résultats de l'agent réactif, bien que parfois les intervalles de confiance soient trop grands pour pouvoir conclure la supériorité stricte. Un plus grand nombre de simulations permettrait de réduire l'intervalle de confiance et s'assurer de la supériorité stricte, cependant la supériorité relative sur l'ensemble des scénarios tend à confirmer la supériorité des algorithmes PR-Plan et YAPPR sur l'agent réactif.

De la même façon, on observe que l'agent avec YAPPR est légèrement supérieur à celui avec PR-Plan. Mais leurs différences sont systématiquement dans l'intervalle de confiance 95%. Il semble donc que YAPPR se comporte mieux que PR-Plan, mais sans certitude. Plusieurs éléments peuvent expliquer cette présumée supériorité de YAPPR. La vitesse d'exécution de YAPPR lors de chaque nouvelle observation peut lui permettre de réagir plus vite et expliquer cette meilleure performance.

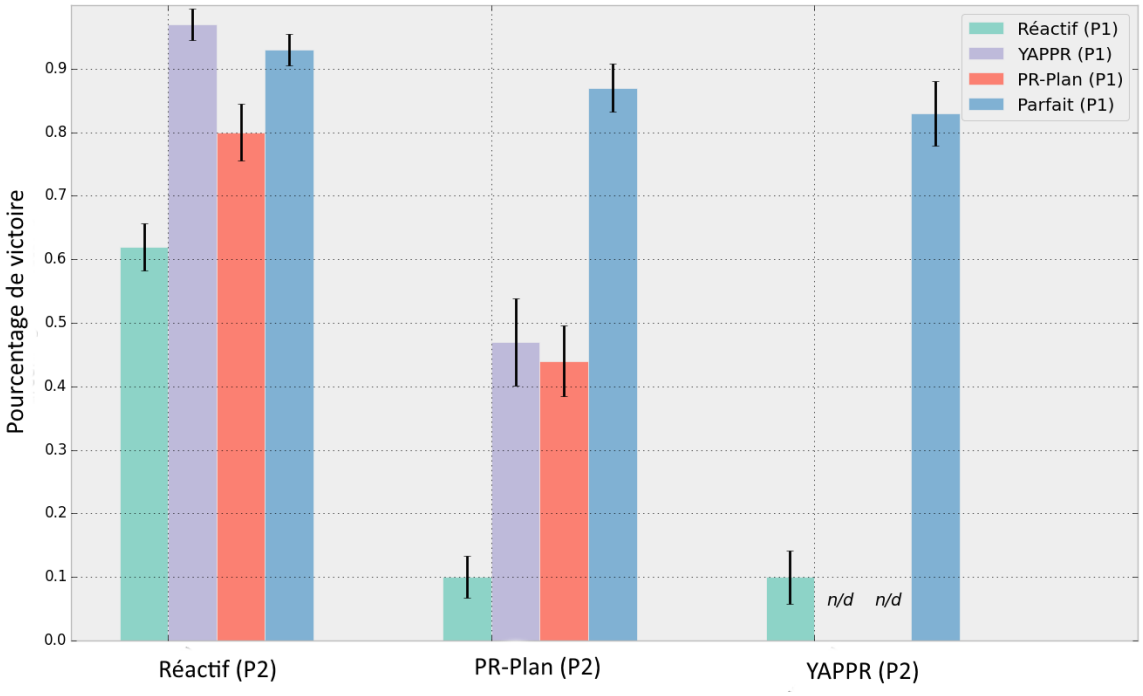

figure 4.5 – Taux de victoire sur la carte *Symmetric Small*

4.3.3 Scénario 2 : *Symmetric Small*

La carte utilisée pour le second scénario est une carte créée à l'aide de l'éditeur fourni avec le jeu StarCraft. Nous avons nommé la carte *Symmetric Small*. La figure 4.4 représente la carte telle que perçue par notre agent. Les points verts représentent les bases du joueur $P1$ et les points rouges les bases du joueur $P2$. Dans ce scénario chaque joueur possède deux groupes de deux unités qui sont répartis dans deux régions.

Nous avons exécuté un total de 3054 parties, toutes paires d'agents confondues, sur ce scénario, soit environ huit jours de simulation continue. La figure 4.5 présente le taux de victoire obtenu par chacun des agents selon leurs opposants.

On observe que de façon similaire au scénario 1, l'agent avec reconnaissance parfaite domine largement les parties avec un taux de victoire contre chaque agent d'au moins 80%. L'agent réactif possède un taux de victoire qui ne dépasse pas les 10%

contre les agents avec les autres algorithmes.

De façon plus prononcée que dans le scénario 1, on remarque que les agents avec les algorithmes YAPPR et PR-Plan sont nettement supérieurs à l'agent réactif. Il n'y a plus de chevauchement d'intervalle de confiance, la différence est même d'au moins 10%.

On remarque toujours un certain avantage pour l'agent utilisant YAPPR par rapport à celui utilisant PR-Plan, mais cet avantage semble se résorber, en particulier quand ils s'affrontent.

Le résultat de YAPPR contre l'agent réactif semble meilleur que celui de l'agent parfait. Il est possible d'expliquer cela par le fait que dans ce scénario précis les règles de l'agent ne sont pas optimales et ainsi YAPPR tombe dans un scénario plus optimal en détectant le plan de l'adversaire plus tard que l'agent parfait.

Cependant, malgré cette anomalie cela n'invalide pas notre agent, car on se rend compte que dans l'ensemble les résultats sont cohérents. Par contre, cela tend à prouver la nécessité de faire des tests sur plusieurs scénarios différents afin de diminuer l'impact de ces anomalies et avoir une comparaison des plus précises.

4.3.4 Discussion

Nos deux simulations montrent que notre système de comparaison via l'affrontement entre deux agents contenant un algorithme de reconnaissance de plan semble avoir du sens. On remarque bien que les algorithmes que nous avons comparés ne sont pas optimaux, mais bien plus efficaces qu'un agent purement réactif. On constate aussi que les résultats comportent quelques variations selon le scénario, mais les tendances sont similaires. Cela nous conforte dans le fait qu'il n'est pas requis de faire des simulations sur des centaines de cartes différentes. La complexité des scénarios est plutôt faible, pourtant on observe tout de même une différence entre les algorithmes. Malheureusement, cette différence est souvent encore dans l'intervalle de confiance de 95%, ce qui empêche toute conclusion catégorique.

Nous avons aussi conservé d'autres métriques de ces simulations qui sont dans les annexes B et C. Ces métriques nous permettent une meilleure compréhension des scénarios et confirme les résultats obtenus en terme de taux de victoire.

On constate avec ces chiffres que YAPPR est plus conservateur en terme du nombre d'unités qui survivent que PR-Plan, ce qui est un indicateur que dans notre environnement de StarCraft YAPPR se comporte mieux que PR-Plan.
Cependant, il est important de nuancer ces observations. Le fait que YAPPR semble meilleur que PR-Plan par le fait de sa rapidité d'exécution est lié à l'implémentation du planificateur utilisé. Pour nos simulations, nous avons utilisé les implémentations existantes et elles n'ont pas le même niveau d'optimisation pour résoudre les problèmes. En effet, PR-Plan est moins mature que YAPPR qui a évolué depuis PHATT. Ainsi une optimisation du temps de calcul de PR-Plan donnerait sûrement de meilleurs résultats.
Néanmoins, les scénarios ont permis de montrer le comportement des algorithmes lors de l'augmentation de la complexité du scénario. Selon ce point de vue, on constate que PR-Plan semble avoir plus de difficulté à suivre que YAPPR.

Chapitre 5

Idées pour des améliorations futures

Le travail effectué pour ce mémoire ne constitue qu'un début dans la comparaison de systèmes de reconnaissance de plan. Ce travail se concentre sur des algorithmes dits de reconnaissance de plan, mais il pourrait être étendu aux modèles de traitement des événements complexes, ou tout autre système qui veut faire de la reconnaissance de plan ou d'intentions. Lors de notre recherche, nous avons fait ce qu'il fallait pour avoir des résultats intéressants, mais il est possible d'imaginer de nombreuses idées pour enrichir et poursuivre dans ce domaine. Nous avons extrait cinq axes d'améliorations. Le premier est une amélioration des scénarios, le deuxième, une amélioration de l'agent lui-même, le troisième, une amélioration du système de comparaison, le quatrième, l'ajout de nouveaux algorithmes à comparer, et le dernier, l'application de notre concept de comparaison sur d'autres domaines, en particulier les activités de la vie quotidienne.

5.1 Amélioration des scénarios

L'amélioration majeure des scénarios serait de pouvoir augmenter la complexité du domaine de reconnaissance de plan. En effet, il serait envisageable d'ajouter progressivement plusieurs éléments du jeu complet dans les scénarios jusqu'à faire

un agent complet. Par exemple, l'ajout de l'observabilité partielle serait un grand avancement. Tout d'abord, il permettrait de tester les algorithmes qui supportent la vision partielle pour voir si les résultats sont semblables à ceux sans vision partielle. Cependant, l'amélioration du scénario est une tâche complexe, car elle nécessite l'amélioration de l'agent pour pouvoir utiliser les nouveaux comportements. Mais il faut aussi améliorer l'agent pour qu'il soit capable de générer de nouvelles observations, créer de nouveaux plans possibles, et améliorer le système de prise de décision afin qu'il puisse utiliser ces nouveaux plans. Cela nécessiterait aussi d'adapter les modèles pour les algorithmes de reconnaissance de plan afin qu'ils puissent reconnaître les nouveaux plans.
Faire un agent intelligent complet basé sur la reconnaissance de plan est intéressant, car cela permettrait de comparer un tel agent avec un autre agent intelligent faisant fi de la reconnaissance de plan. Cela confirmerait ou infirmerait la pertinence de la reconnaissance de plan dans ce contexte.

5.2 Amélioration du système de prise de décision

Actuellement, le système de prise de décision est plutôt simple. On a vu que les résultats étaient tout de même pertinents, cependant ce système de prise de décision ne semble pas efficace à long terme, en particulier si l'on veut enrichir les scénarios de nouveaux comportements. L'utilisation de connaissances expertes et l'analyse empirique sont acceptables dans la mesure où le domaine est simple. Aussi, il serait bon que le système de prise de décision n'utilise plus uniquement le plan de plus haute probabilité comme seule information pertinente, mais qu'il considère aussi la probabilité des autres plans afin de pouvoir prendre de meilleures décisions. La comparaison des algorithmes de reconnaissance de plan serait plus complète, car on jugerait non seulement le plus probable, mais aussi l'ensemble des hypothèses.

5.3 Amélioration du système de comparaison

L'amélioration du système de comparaison aurait pour but de le rendre plus standard et facilement réutilisable. En effet, la mise en place de notre infrastructure

pour nos simulations était complexe et longue. Le fait d'utiliser des technologies mieux établies pourrait aider toute personne cherchant à développer un algorithme de reconnaissance de plan, lui permettant de l'intégrer, de le tester et de le comparer aux autres algorithmes simplement.

L'implémentation actuelle utilise une communication par socket TCP pure avec des messages précis et non documentés, ce qui rend le développement par de tierces personnes difficiles. Il serait possible d'utiliser des systèmes de communication mieux reconnus, tel que Java Messaging Service (JMS) avec des messages XML. Cela donnerait une interface de communication standard entre l'agent et les systèmes de reconnaissance de plan, d'autant que JMS comporte des implémentations dans la plupart des langages de programmation ce qui ne contraindrait pas le développement d'algorithme à un langage spécifique.

Ensuite, actuellement la détection d'abandon de plan est fortement liée à l'agent. Cela pourrait être découplé et externalisé un peu comme pour la reconnaissance de plan. Ainsi, il serait possible de développer des systèmes de détection d'abandon de plan. Il serait possible de comparer les systèmes de détection d'abandon de plan de façon analogue à ce qu'on a fait dans ce mémoire. Par contre, dans ce cas il est important d'utiliser les mêmes agents de reconnaissance de plan afin de ne pas biaiser cette comparaison.

5.4 Ajout de nouveaux algorithmes

Nous avons fait nos expérimentations avec deux algorithmes de reconnaissance de plan différents. Mais il serait intéressant de faire des tests avec plusieurs autres algorithmes afin de pouvoir faire une classification et une analyse plus exhaustive de ceux-là. Cette tâche peut aussi être faite par les chercheurs désireux de comparer de nouveaux algorithmes à ceux déjà existants.

5.5 Expérimentations sur les activités de la vie quotidienne

La reconnaissance d'activités de la vie quotidienne est un domaine de recherche active. Plusieurs articles sont publiés par l'un des pionniers de la reconnaissance de plan, Kautz, avec Liao et Fox en 2007 [13] et avec Sadilek en 2010 [17], où ils utilisent les données GPS pour reconnaître les activités.

Le laboratoire du DOMUS avec son appartement intelligent permet de faire des simulations et cela peut être vraiment utile pour comparer les algorithmes. De plus, Chikhaoui, Wang et Pigot ont développé une approche pour la reconnaissance d'activité [5, 4] et créé un ensemble de données pour tester leur approche.

Alors il serait intéressant de modéliser le domaine des activités de la vie quotidienne pour voir quels algorithmes de reconnaissance de plan sont efficaces. De plus, faire des tests sur un tel domaine aurait l'avantage de voir l'influence du domaine sur les algorithmes de reconnaissance de plan et ainsi voir leur polyvalence. Par exemple, dans le cas de StarCraft la reconnaissance de plan la plus immédiate possible est nécessaire alors que dans le cas des activités de la vie quotidienne, la vitesse d'exécution des tâches est plus lente et ainsi l'immédiateté du résultat est moins importante que son exactitude.

Donc faire des expérimentations sur les activités de la vie quotidienne pourrait nous apprendre deux éléments intéressants. Tout d'abord, si les algorithmes de reconnaissance de plan s'appliquent aussi pour la reconnaissance d'activités, c'est-à-dire confirmer leur potentiel de généralisation. Ensuite, la comparaison des algorithmes serait plus que pertinente. On pourrait ainsi voir si un algorithme de reconnaissance de plan est meilleur, peu importe le domaine, ou si selon le domaine certains algorithmes ou familles d'algorithmes se comportent mieux. Ainsi l'expérimentation sur un autre domaine sensiblement différent améliorerait la validité de la comparaison entre les algorithmes de reconnaissance de plan.

Finalement, le fait que le DOMUS possède un appartement intelligent pourrait permettre de valider les résultats dans un scénario réel.

Conclusion

La reconnaissance de plan est un problème complexe où l'on cherche à reconnaître le plan et les intentions d'un agent. Ce domaine de recherche ne comporte pas de processus de comparaison entre les divers algorithmes de façon simple et objective. Pour cette raison, nous avons décidé d'établir un processus pour faire la comparaison entre les divers algorithmes de reconnaissance de plan. Pour ce faire, nous avons décidé de faire une comparaison des deux principaux algorithmes à l'aide d'une observation *in situ* dans le jeu StarCraft.

Nous avons donc créé un agent qui utilise les plans reconnus de l'adversaire dans son processus de décision. Puis, nous avons préparé deux scénarios dans lesquels nous avons fait évoluer nos agents. Enfin, nous avons exécuté de nombreuses simulations avec les divers algorithmes afin de pouvoir évaluer notre environnement de comparaison et de voir quel algorithme est le meilleur.

L'analyse des résultats nous a permis de nous assurer que notre hypothèse était valide, et donc de confirmer que notre méthodologie de comparaison était valide et prometteuse pour de futures comparaisons. Nos résultats nous ont permis de montrer que YAPPR se comporte mieux sur les scénarios, mais avec une différence qui n'est pas significative avec PR-Plan.

Cependant, il existe encore plusieurs améliorations possibles pour faire des comparaisons plus exhaustives, mais le travail effectué pose les bases pour des comparaisons futures entre divers algorithmes de reconnaissance de plan.

L'amélioration de ce système de comparaison est pertinente pour l'évolution des algorithmes de reconnaissance de plan puisqu'il peut assister les chercheurs à voir les forces et les faiblesses de leurs algorithmes en plus de fournir un environnement de comparaison complexe et complet pour ces algorithmes.

Annexe A

Résultats détaillés pour les scénarios sur des cartes aléatoires

Ci-dessous sont les matrices de confusion et les valeurs de précision, rappel, exactitude, score F1 et coefficient Kappa (κ) des expérimentations sur les cartes aléatoires effectuées lors du chapitre 4.
Nous avons généré 50 cartes en utilisant le modèle de Erdős et Rényi [6], chacune contenant 50 régions. Les régions possèdent un facteur de branchement moyen de 3.5. Pour chacune des cartes générées, nous avons déterminé un point de départ et cinq destinations possibles. Puis sur chaque carte nous avons créé un scénario pour chaque point de départ et destination possible. Si plusieurs chemins de même longueur existent, nous créons un scénario pour chaque chemin.

Scénario	Vrais Positifs	Faux Négatifs	Faux Positifs	Vrais Négatifs	Temps (en ms)
10%	171	221	1041	527	6.25
30%	198	194	806	762	6.25
50%	265	127	455	1113	2.84
70%	332	60	255	1313	1.77
100%	392	0	63	1505	1.03

tableau A.1 – Matrice de confusion avec YAPPR

Scénario	Précision	Rappel	Exactitude	Score F1	κ
10%	0.14	0.44	0.36	0.21	0.20
30%	0.20	0.51	0.49	0.28	0.36
50%	0.37	0.68	0.70	0.48	0.63
70%	0.57	0.85	0.84	0.68	0.80
100%	0.86	1.00	0.97	0.93	0.96

tableau A.2 – Résultats avec YAPPR

Scénario	Vrais Positifs	Faux Négatifs	Faux Positifs	Vrais Négatifs	Temps (en ms)
10%	392	0	1421	147	42 798.58
30%	321	71	1009	559	42 798.58
50%	325	67	498	1070	64 401.88
70%	366	26	287	1281	77 042.64
100%	391	1	79	1489	92 623.48

tableau A.3 – Matrice de confusion avec PR-Plan

Scénario	Précision	Rappel	Exactitude	Score F1	κ
10%	0.22	1.00	0.28	0.36	0.09
30%	0.24	0.82	0.45	0.37	0.31
50%	0.39	0.83	0.71	0.53	0.64
70%	0.56	0.93	0.84	0.70	0.80
100%	0.83	1.00	0.96	0.91	0.95

tableau A.4 – Résultats avec PR-Plan

Annexe B

Résultats détaillés des scénarios sur la carte *Symmetric Small*

Dans cette annexe nous présentons les résultats détaillés de nos simulations sur la carte *Symmetric Small* (figure B.1) avec les agents *in situ*.
Le tableau B.1 présente le temps moyen en *frame* pour chaque affrontement, le Taux de victoire du joueur 1, l'intervalle de confiance et le nombre total de parties jouées.
Les tableaux B.2 et B.3 présentent des statistiques avancées de ces parties, telles que le nombre moyen de bâtiments perdus par partie, la santé moyenne restante des bâtiments à la fin des scénarios et le nombre d'unités restantes à la fin du scénario.

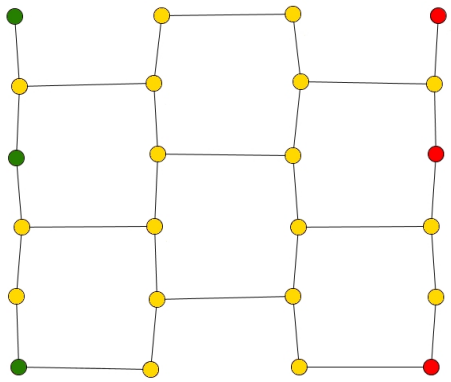

figure B.1 – Carte *Symmetric Small*

J1	J2	*Frame*	Victoires	Parties totales	Confiance (95%)
Reactif	Reactif	2 362,98	0,62	650,00	0,037
Perfect	Reactif	2 078,06	0,93	395,00	0,025
YAPPR	Reactif	2 369,01	0,97	200,00	0,025
PR-Plan	Reactif	2 447,55	0,80	301,00	0,045
Reactif	PR-Plan	2 226,35	0,10	302,00	0,033
Perfect	PR-Plan	2 371,82	0,87	300,00	0,038
YAPPR	PR-Plan	2 421,59	0,47	200,00	0,069
PR-Plan	PR-Plan	2 509,79	0,44	300,00	0,056
Reactif	YAPPR	2 431,01	0,10	200,00	0,042
Perfect	YAPPR	2 461,38	0,83	206,00	0,051

tableau B.1 – Taux de victoire du joueur 1 sur la carte *Symmetric Small*

J1	J2	Bâtiments perdus (en %)	Santé restante (en %)	Unités restantes
Reactif	Reactif	0,03	0,95	1,06
Perfect	Reactif	0,01	0,98	3,33
YAPPR	Reactif	0,09	0,96	3,10
PR-Plan	Reactif	0,07	0,95	1,70
Reactif	PR-Plan	0,00	0,97	0,16
Perfect	PR-Plan	0,02	0,98	3,14
YAPPR	PR-Plan	0,00	0,98	2,88
PR-Plan	PR-Plan	0,01	0,97	1,18
Reactif	YAPPR	0,01	0,96	0,17
Perfect	YAPPR	0,00	0,99	3,33

tableau B.2 – Détails pour le joueur 1 sur la carte *Symmetric Small*

J1	J2	Bâtiments perdus (en %)	Santé restante (en %)	Unités restantes
Reactif	Reactif	0,04	0,94	0,61
Perfect	Reactif	0,00	0,97	0,11
YAPPR	Reactif	0,01	0,96	0,07
PR-Plan	Reactif	0,01	0,96	1,47
Reactif	PR-Plan	0,01	0,97	2,66
Perfect	PR-Plan	0,02	0,97	0,33
YAPPR	PR-Plan	0,00	0,98	2,63
PR-Plan	PR-Plan	0,00	0,98	1,53
Reactif	YAPPR	0,13	0,95	3,12
Perfect	YAPPR	0,00	0,98	0,54

tableau B.3 – Détails pour le joueur 2 sur la carte *Symmetric Small*

Annexe B. Résultats détaillés des scénarios sur la carte *Symmetric Small*

Annexe C

Résultats détaillés des scénarios sur la carte *Astral Balance*

Dans cette annexe nous présentons les résultats détaillés de nos simulations sur la carte *Astral Balance* (figure C.1) avec les agents *in situ*.
Le tableau C.1 présente le temps moyen en *frame* pour chaque affrontement, le taux de victoire du joueur 1, l'intervalle de confiance et le nombre total de parties jouées.
Les tableaux C.2 et C.3 présentent des statistiques avancées de ces parties, telles que le nombre moyen de bâtiments perdus par partie, la santé moyenne restante des bâtiments à la fin des scénarios et le nombre d'unités restantes à la fin du scénario.

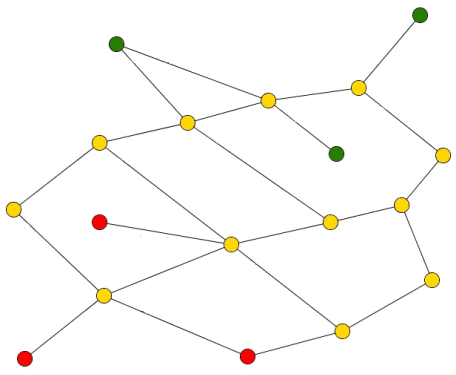

figure C.1 – Carte *Astral Balance*

J1	J2	*Frame*	Victoires	Parties totales	Confiance (95%)
Reactif	Reactif	3 054	0,39	310	0,054
Perfect	Reactif	2 627	0,86	302	0,039
YAPPR	Reactif	3 313	0,66	100	0,093
PR-Plan	Reactif	2 967	0,57	302	0,056
Reactif	PR-Plan	2 692	0,17	268	0,045
Perfect	PR-Plan	2 671	0,70	271	0,055
YAPPR	PR-Plan	3 508	0,36	154	0,076
PR-Plan	PR-Plan	2 629	0,25	150	0,069
Reactif	YAPPR	2 963	0,12	50	0,090
Perfect	YAPPR	2 995	0,65	100	0,093
PR-Plan	YAPPR	3 134	0,19	53	0,105

tableau C.1 – Taux de victoire du joueur 1 sur la carte *Astral Balance*

J1	J2	Bâtiments perdus (en %)	Santé restante (en %)	Unités restantes
Reactif	Reactif	0,29	0,86	0,31
Perfect	Reactif	0,05	0,94	3,15
YAPPR	Reactif	0,16	0,89	3,08
PR-Plan	Reactif	0,28	0,88	1,94
Reactif	PR-Plan	0,10	0,91	0,42
Perfect	PR-Plan	0,03	0,97	2,35
YAPPR	PR-Plan	0,77	0,82	0,97
PR-Plan	PR-Plan	0,03	0,94	0,58
Reactif	YAPPR	0,14	0,92	0,17
Perfect	YAPPR	0,02	0,96	3,33
PR-Plan	YAPPR	0,02	0,92	3,33

tableau C.2 – Détails pour le joueur 1 sur la carte *Astral Balance*

J1	J2	Bâtiments perdus (en %)	Santé restante (en %)	Unités restantes
Reactif	Reactif	0,05	0,90	1,72
Perfect	Reactif	0,02	0,93	0,31
YAPPR	Reactif	0,41	0,84	0,78
PR-Plan	Reactif	0,07	0,90	2,77
Reactif	PR-Plan	0,05	0,93	2,78
Perfect	PR-Plan	0,04	0,94	0,83
YAPPR	PR-Plan	1,04	0,76	2,16
PR-Plan	PR-Plan	0,04	0,95	2,77
Reactif	YAPPR	0,16	0,92	3,02
Perfect	YAPPR	0,35	0,90	0,97
PR-Plan	YAPPR	0,19	0,93	3,26

tableau C.3 – Détails pour le joueur 2 sur la carte *Astral Balance*

Annexe C. Résultats détaillés des scénarios sur la carte *Astral Balance*

Annexe D

Exemple de domaine PR-Plan pour le scénario 1

programme D.1 – domain.pddl

```
1 ;; StarCraft domain Typed version.
2
3 (define (domain starcraft)
4   (:requirements :strips :typing)
5   (:types location units - object
6           fighter builder - units
7   )
8   (:predicates
9         (at ?obj - units ?loc - location)
10        (link ?loc1 - location ?loc2 - location)
11  )
12  (:action MOVE-UNIT
13   :parameters (?unit - units ?loc_from - location ?loc_to - location)
14   :precondition (and   (not (= ?loc_from ?loc_to))
15                        (at ?unit ?loc_from)
16                        (link ?loc_from ?loc_to))
17   :effect (and (not (at ?unit ?loc_from)) (at ?unit ?loc_to))
18  )
19 )
```

programme D.2 – hyps.dat

```
(at unit pos3)
(at unit pos1)
(at unit pos5)
(at unit pos12)
(at unit pos16)
(at unit pos18)
```

programme D.3 – obs.txt

```
(MOVE-UNIT unit pos3 pos2)
```

programme D.4 – template.pddl

```
(define (problem Astral_Balance)
  (:domain starcraft)
  (:objects
    pos3 pos4 pos1 pos2 pos7 pos8 pos5 pos6 pos11 pos12
    pos9 pos10 pos15 pos16 pos13 pos14 pos18 pos17 - location
    unit - fighter
  )
  (:init
    (link pos3 pos2)(link pos3 pos4)(link pos4 pos6)(link pos7 pos6)
    (link pos4 pos5)(link pos2 pos8)(link pos4 pos3)(link pos2 pos3)
    (link pos2 pos6)(link pos17 pos4)(link pos4 pos17)
    (link pos7 pos15)(link pos7 pos11)(link pos1 pos13)
    (link pos8 pos2)(link pos8 pos11)(link pos5 pos4)
    (link pos6 pos16)(link pos6 pos10)(link pos6 pos2)
    (link pos6 pos4)(link pos6 pos7)(link pos11 pos8)
    (link pos11 pos9)(link pos11 pos7)(link pos12 pos14)
    (link pos9 pos13)(link pos9 pos11)(link pos10 pos6)
    (link pos10 pos15)(link pos10 pos17)(link pos15 pos18)
    (link pos15 pos14)(link pos15 pos10)(link pos15 pos7)
    (link pos16 pos6)(link pos13 pos14)(link pos13 pos1)
    (link pos13 pos9)(link pos14 pos18)(link pos14 pos15)
    (link pos14 pos12)(link pos14 pos13)(link pos18 pos15)
    (link pos18 pos14)(link pos17 pos10)(at unit pos3)
  )
  (:goal (and <HYPOTHESIS>) )
)
```

Annexe E

Exemple de domaine YAPPR pour le scénario 1

programme E.1 – domain.lisp

```
1 (import 'sift.planrec-htn-framework::make-term-rule)
2 (import 'sift.planrec-htn-framework::full-ord-and-rule)
3 (import 'sift.planrec-htn-framework::or-rule)
4 (import 'sift.planrec-htn-framework::unord-and-rule)
5
6 ;=============================================
7 ;Generated Library for test
8 ;=============================================
9
10 ; base actions
11 (setf unit-at-1-action
12         (make-term-rule +demo+ 'unit-at-1-action 'unit-at-1))
13 (setf unit-at-10-action
14         (make-term-rule +demo+ 'unit-at-10-action 'unit-at-10))
15 (setf unit-at-11-action
16         (make-term-rule +demo+ 'unit-at-11-action 'unit-at-11))
17 (setf unit-at-12-action
18         (make-term-rule +demo+ 'unit-at-12-action 'unit-at-12))
19 (setf unit-at-13-action
20         (make-term-rule +demo+ 'unit-at-13-action 'unit-at-13))
21 (setf unit-at-14-action
```

```
22          (make-term-rule +demo+ 'unit-at-14-action 'unit-at-14))
23 (setf unit-at-15-action
24          (make-term-rule +demo+ 'unit-at-15-action 'unit-at-15))
25 (setf unit-at-16-action
26          (make-term-rule +demo+ 'unit-at-16-action 'unit-at-16))
27 (setf unit-at-17-action
28          (make-term-rule +demo+ 'unit-at-17-action 'unit-at-17))
29 (setf unit-at-18-action
30          (make-term-rule +demo+ 'unit-at-18-action 'unit-at-18))
31 (setf unit-at-2-action
32          (make-term-rule +demo+ 'unit-at-2-action 'unit-at-2))
33 (setf unit-at-3-action
34          (make-term-rule +demo+ 'unit-at-3-action 'unit-at-3))
35 (setf unit-at-4-action
36          (make-term-rule +demo+ 'unit-at-4-action 'unit-at-4))
37 (setf unit-at-5-action
38          (make-term-rule +demo+ 'unit-at-5-action 'unit-at-5))
39 (setf unit-at-6-action
40          (make-term-rule +demo+ 'unit-at-6-action 'unit-at-6))
41 (setf unit-at-7-action
42          (make-term-rule +demo+ 'unit-at-7-action 'unit-at-7))
43 (setf unit-at-8-action
44          (make-term-rule +demo+ 'unit-at-8-action 'unit-at-8))
45 (setf unit-at-9-action
46          (make-term-rule +demo+ 'unit-at-9-action 'unit-at-9))
47
48 ; Move plans
49 (setf move-from-2-to-1-plan
50     (full-ord-and-rule +demo+ 'move-from-2-to-1-plan
51          'unit-at-2-action 'unit-at-9-action 'unit-at-13-action
52          'unit-at-8-action 'unit-at-15-action 'unit-at-1-action))
53 (setf move-from-2-to-11-plan
54     (full-ord-and-rule +demo+ 'move-from-2-to-11-plan
55          'unit-at-2-action 'unit-at-7-action 'unit-at-11-action))
56 (setf move-from-2-to-14-plan
57     (or-rule +demo+ 'move-from-2-to-14-plan
58          'plan5-plan 'plan6-plan))
59 (setf move-from-2-to-3-plan
60     (full-ord-and-rule +demo+ 'move-from-2-to-3-plan
```

```
61          'unit-at-2-action 'unit-at-3-action))
62 (setf move-from-2-to-5-plan
63     (or-rule +demo+ 'move-from-2-to-5-plan
64          'plan1-plan 'plan2-plan))
65 (setf move-from-2-to-6-plan
66     (or-rule +demo+ 'move-from-2-to-6-plan
67          'plan3-plan 'plan4-plan))
68 (setf plan1-plan
69     (full-ord-and-rule +demo+ 'plan1-plan
70          'unit-at-2-action 'unit-at-3-action 'unit-at-4-action
71          'unit-at-5-action))
72 (setf plan2-plan
73     (full-ord-and-rule +demo+ 'plan2-plan
74          'unit-at-2-action 'unit-at-7-action 'unit-at-4-action
75          'unit-at-5-action))
76 (setf plan3-plan
77     (full-ord-and-rule +demo+ 'plan3-plan
78          'unit-at-2-action 'unit-at-7-action 'unit-at-12-action
79          'unit-at-18-action 'unit-at-6-action))
80 (setf plan4-plan
81     (full-ord-and-rule +demo+ 'plan4-plan
82          'unit-at-2-action 'unit-at-7-action 'unit-at-10-action
83          'unit-at-18-action 'unit-at-6-action))
84 (setf plan5-plan
85     (full-ord-and-rule +demo+ 'plan5-plan
86          'unit-at-2-action 'unit-at-7-action 'unit-at-12-action
87          'unit-at-18-action 'unit-at-16-action 'unit-at-14-action))
88 (setf plan6-plan
89     (full-ord-and-rule +demo+ 'plan6-plan
90          'unit-at-2-action 'unit-at-7-action 'unit-at-10-action
91          'unit-at-18-action 'unit-at-16-action 'unit-at-14-action))
92
93 ; Goals
94 (setf rts-goals '(move-from-2-to-14-plan move-from-2-to-6-plan
95                   move-from-2-to-1-plan move-from-2-to-11-plan
96                   move-from-2-to-5-plan move-from-2-to-3-plan ))
97
98 ; Plans
99 (setf +rts-plan-library+
```

```
100         (sift.planrec-framework::compile-plan-library
101                 +demo+
102                 :rules
103                 (list unit-at-1-action unit-at-10-action
104                     unit-at-11-action unit-at-12-action
105                     unit-at-13-action unit-at-14-action
106                     unit-at-15-action unit-at-16-action
107                     unit-at-17-action unit-at-18-action
108                     unit-at-2-action unit-at-3-action
109                     unit-at-4-action unit-at-5-action unit-at-6-action
110                     unit-at-7-action unit-at-8-action unit-at-9-action
111                     move-from-2-to-1-plan move-from-2-to-11-plan
112                     move-from-2-to-14-plan move-from-2-to-3-plan
113                     move-from-2-to-5-plan move-from-2-to-6-plan
114                     plan1-plan plan2-plan plan3-plan plan4-plan
115                     plan5-plan plan6-plan)
116                 :intendable-head-symbols rts-goals))
117
118 ; Utils
119 (defun rts-start-state ()
120   (sift.planrec-framework::initial-explanations
121     +demo+ rts-goals +rts-plan-library+))
122
123 (setf rts-start-state (rts-start-state))
124
125 (defun rts-do-next (state obs)
126   (sift.planrec-framework::step-actions
127     +demo+ state obs +rts-plan-library+))
```

Bibliographie

[1] Francis Bisson.
« La reconnaissance de plan des adversaires ».
Mémoire de maîtrise, Université de Sherbrooke, 2012.

[2] Sviatoslav Braynov.
« Adversarial planning and plan recognition : Two sides of the same coin ».
Dans *Secure Knowledge Management Workshop*, 2006.

[3] Eugene Charniak et Robert P Goldman.
« A Bayesian model of plan recognition ».
Artificial Intelligence, 64(1):53–79, 1993.

[4] Belkacem Chikhaoui.
« *Une approche basée sur l'analyse des séquences pour la reconnaissance des activités et comportements dans les environnements intelligents* ».
Thèse de doctorat, 2013.

[5] Belkacem Chikhaoui, Shengrui Wang et Helene Pigot.
« A new algorithm based on sequential pattern mining for person identification in ubiquitous environments ».
Dans *KDD Workshop on Knowledge Discovery from Sensor Data*, pages 19–28, 2010.

[6] Paul Erdős et Alfréd Rényi.
« On random graphs ».
Publicationes Mathematicae Debrecen, 6:290–297, 1959.

[7] Christopher W Geib et Robert P Goldman.
« A probabilistic plan recognition algorithm based on plan tree grammars ».

Artificial Intelligence, 173(11):1101–1132, 2009.

[8] Christopher W Geib et Robert P Goldman.
« Handling looping and optional actions in YAPPR. ».
Plan, Activity, and Intent Recognition, 2010.

[9] Christopher W Geib, John Maraist et Robert P Goldman.
« A new probabilistic plan recognition algorithm Based on String Rewriting. ».
Dans *ICAPS*, pages 91–98, 2008.

[10] Froduald Kabanza, Julien Filion, Abder Rezak Benaskeur et Hengameh Irandoust.
« Controlling the hypothesis space in probabilistic plan recognition ».
Dans *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 2306–2312. AAAI Press, 2013.

[11] Henry A Kautz et James F Allen.
« Generalized plan recognition. ».
Dans *AAAI*, volume 86, pages 32–37, 1986.

[12] J Richard Landis et Gary G Koch.
« The measurement of observer agreement for categorical data ».
Biometrics, pages 159–174, 1977.

[13] Lin Liao, Dieter Fox et Henry Kautz.
« Extracting places and activities from GPS traces using hierarchical conditional random fields ».
The International Journal of Robotics Research, 26(1):119–134, 2007.

[14] Viliam Lisỳ, Radek Píbil, Jan Stiborek, Branislav Bosanskỳ et Michal Pechoucek.
« Game-theoretic approach to adversarial plan recognition ».
Dans *ECAI*, pages 546–551, 2012.

[15] Miquel Ramirez et Hector Geffner.
« Probabilistic plan recognition using off-the-shelf classical planners ».
Dans *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*, pages 1121—1126.

[16] Miquel Ramirez et Hector Geffner.
« Plan recognition as planning ».
Dans *Proceedings of the 21st International Joint Conference on Artificial intelligence. Morgan Kaufmann Publishers Inc*, pages 1778–1783, 2009.

[17] Adam Sadilek et Henry A Kautz.
« Recognizing Multi-Agent Activities from GPS Data. ».
Dans *AAAI*, volume 39, page 109, 2010.

[18] Charles F. Schmidt, N.S. Sridharan et John L. Goodson.
« The plan recognition problem : An intersection of psychology and artificial intelligence ».
Artificial Intelligence, 11(1):45–83, 1978.

[19] Ben George Weber et Michael Mateas.
« A data mining approach to strategy prediction ».
Dans *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 140–147. IEEE, 2009.