Opponent Behaviour Recognition for Real-Time Strategy Games

Froduald Kabanza and **Philipe Bellefeuille** and **Francis Bisson**
Université de Sherbrooke
Sherbrooke, QC J1K 2R1, Canada
{kabanza, philipe.bellefeuille,francis.bisson}@usherbrooke.ca

Abder Rezak Benaskeur and **Hengameh Irandoust**
Defence R&D Canada - Valcartier
Québec, QC G3J 1X5, Canada
{abderrezak.benaskeur, hengameh.irandoust}@drdc-rddc.gc.ca

Abstract

In Real-Time Strategy (RTS) video games, players (controlled by humans or computers) build structures and recruit armies, fight for space and resources in order to control strategic points, destroy the opposing force and ultimately win the game. Players need to predict where and how the opponents will strike in order to best defend themselves. Conversely, assessing how the opponents will defend themselves is crucial to mounting a successful attack while exploiting the vulnerabilities in the opponent's defence strategy. In this context, to be truly adaptable, computer-controlled players need to recognize their opponents' behaviour, their goals, and their plans to achieve those goals. In this paper we analyze the algorithmic challenges behind behaviour recognition in RTS games and discuss a generic RTS behaviour recognition system that we are developing to address those challenges. The application domain is that of RTS games, but many of the key points we discuss also apply to other video game genres such as multiplayer first person shooter (FPS) games.

Introduction

Year 2735: the human race has colonized a distant area of the galaxy. Under the threat of an aggressive alien race called the Zerg, the colonies have united under the banner of the Terran Dominion. The war against the Zerg has been very taxing and we are in dire need of natural resources. A new hospitable planet was just discovered and a small force, with you as their leader, was sent to take control of it. Unfortunately, as your ship orbits the newly found planet prior to landing, initial readings tell you that the Zerg were one step ahead and have just settled a small colony on the larger continent. You decide to land your army on that same continent to destroy the enemy's colony while it's still young and assert control of the new planet.

The small army you brought with you will not be enough, however; you will need reinforcements. Unfortunately, because of the resource problem the Terran Dominion is already facing, you are expected to gather the resources needed to gear your men and build your structures. You may even be required to build your own factories to produce warships, tanks and any other war machines you may need. Nonetheless, you are encouraged by the fact that the enemy is in a similar situation and if you can outwit them, predict their moves and stay one step ahead, this planet will soon be under the Terran Dominion's control, a first step towards its victory, and its survival.

The previous paragraphs describe StarCraft I, a real-time strategy (RTS) computer game released in 1998 by Blizzard Entertainment. As with other similar RTS games – such as the Age of Empires series and the Command & Conquer series – StarCraft involves multiple opposing forces, each consisting of a team of one or more players, and each player controlling a given number of army units. The victory condition can be different from one scenario to another, but usually involves destroying some or all the opponents' buildings or controlling strategic points for a certain period of time. Units are typically recruited from structures that the players build using different resources, as the game advances. As armies build up on each side, initial skirmishes can quickly escalate into all-out wars.

Players in RTS games need to be constantly aware of the behaviours of their opponents to evaluate the threat they pose and anticipate the potential threat that may arise in the near future. Correctly predicting the adversary's moves is key to deciding winning moves, whereas incorrectly identifying them often puts the player in a dire position. The infinite creativity of the human mind coupled with its sometimes chaotic and unpredictable nature makes the behaviour recognition of human players a very difficult algorithmic problem for an AI system. Time becomes a very important factor because one wants to predict the adversary's strategy as soon as possible in order to counter it in time. Any information gathered to predict the enemy's strategy is only valid for a certain amount of time. If a player scouted the enemy's army a few minutes ago, he can expect that the army's composition has changed since. If he has not scouted an area in a while, he cannot be sure that it is still free of enemies.

Uncertainty is another important factor. Players only have a partial view of the situation as a whole. A player does not see areas where he does not control any units. He usually

cannot see the units that are being recruited by the enemy, even if he sees the structures in which the unit is being recruited. He does not know the amount of resources his opponent has gathered or the amount he has remaining. In some cases, he may not even know what abilities are available to an opposing unit. The number of strategies available to the opponent adds to the uncertainty as one does not know for certain what his opponent will do next.

The 2010 conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE) will be hosting a StarCraft AI competition as part of the conference program. The objective is to enable academic researchers to evaluate their AI systems by having them compete against each other in the StarCraft environment. We are planning to eventually participate in such competitions. With this objective, behaviour recognition is only one component of a larger AI system which in addition involves components to make decisions on how to act against the opposing force, execute and monitor planned and reactive actions, and learn from past interactions to adapt accordingly.

The architecture of our behaviour recognition system is sketched in Figure 1. The system is named Hostile Intent, Capability and Opportunity Recognizer (HICOR), and consists of three intertwined inferences for recognizing the intent, capability, and opportunity of the opposing force. The intent recognition process takes as input a plan library, game state updates, and an influence map. The influence map is a heuristically computed overlay of the map in the current game state characterizing the current influence of the different units in the game. The output of the intent recognition process is a set of plans recognized as being concurrently pursued by the opposing force and hypothetically consistent with observations from the game state updates. The plans are annotated with the corresponding goals.

In the intent recognition process, some details on the use of resources by actions are abstracted away. It is the role of the capability analysis component to analyze in detail the resources associated to actions in the plan and to determine to what extent the plans are achievable. The output is a refinement of the recognized plan by detailing the involved actions and resources.

Considering that a plan is recognized based on the observations to date, the opportunity analysis component projects the "future part of the recognized plan" into the near future to evaluate its potential success (i.e., opportunities from the opponent's standpoint) and vulnerabilities (i.e., opportunities to counter the plan from the RTS AI system's standpoint). Potential success or vulnerabilities correspond to action preconditions that may become enabled by the environment – not just by the opposing force but also possibly by the own force.

In this paper we focus on the intent recognition component. The opponent intent recognition problem is framed as one of hypothesizing a set of concurrent plans currently being executed and the corresponding current execution status. We adopt the Probabilistic Hostile Agent Task Tracker (PHATT) (Goldman, Geib, and Miller 1999; Geib and Goldman 2009), to which we add three improvements. Firstly, we introduce explicit time and resources,

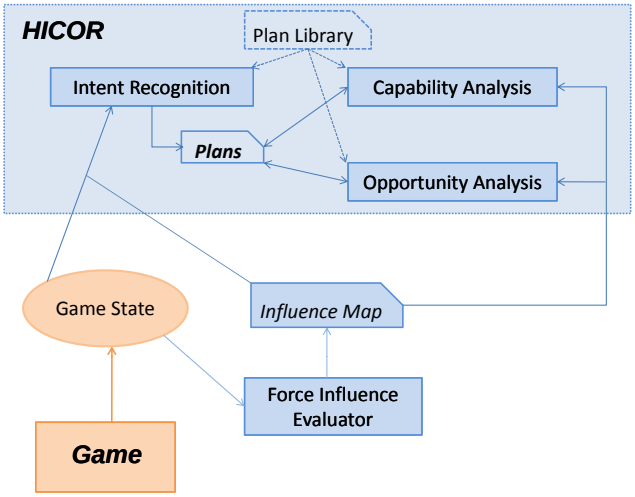

Figure 1: HICOR's high level architecture

which are crucial in the RTS domain for an accurate modelling of intent, capability and opportunity. Secondly, we take into account the current influence of the units composing the opposing force to initialize the prior probabilities for the plan library. This is important because opponents strategize by aiming to cause as much damage as possible, while taking into account the opposing force. Finally, we explicitly model actions used by the agent to gather information helping recognizing the opponent's strategy.

The rest of this paper continues with a brief discussion on related work with a particular emphasis on PHATT from which much of our inspiration is drawn. We then discuss the plan recognition problem within the RTS domain, using scenarios taken from StarCraft. A description of the plan recognition module of HICOR's intent recognition component follows, succeeded by preliminary experiments and a conclusion with future developments.

Related Work

It is natural to approach the problem of recognizing the adversary's behaviour as a plan recognition problem. After all, players (human or artificial) in video games behave by executing plans or similar goal-oriented strategies. For artificial agents, the plans are often implemented as finite state machines (Rabin 2005) – that is, fundamentally predefined goal-driven reaction rules.

Plan recognition has long been recognized as a key reasoning process at the heart of human cognition. In his 1976 psychological experiment, Schmidt provided evidence that humans do infer hypotheses about the plans and goals of other agents and use these hypotheses in subsequent reasoning (Schmidt 1976). Later, with his colleagues Sridharan and Goodson, they positioned plan recognition as a central problem in the design of intelligent systems (Schmidt, Sridharan, and Goodson 1978). Many theories and computational frameworks for plan recognition have since been introduced and experimented with in var-

ious domains (e.g., (Kautz 1991; Avrahami-Zilberbrand and Kaminka 2007; Pynadath and Wellman 1995; Blaylock and Allen 2006; Bui, Venkatesh, and West 2002)).

The recognition of strategies and tactics of opponents in RTS games is a particular case of the adversarial plan recognition problem, in which the observed agent may adopt different behaviours to try to fool the observer. Many adversarial plan recognition frameworks have been proposed and a significant sample of them is collected in (Kott and McEneaney 2007). We embraced PHATT because of its easy implementation, yet being one of the existing approaches providing a higher coverage of the basic features required for RTS games, such as the handling of concurrent plans and being able to model plan abandonment.

PHATT frames the adversarial plan recognition problem into a Hidden Markov Model (HMM). The HMM is not explicitly specified. Instead, it is unfolded on the fly through the simulation of concurrent plans being hypothetically executed by the observed agent. A state of the HMM is a set of concurrent plans the agent may be pursuing and the current status in the execution of these plans. From these current plan execution statuses, the set of actions that could be executed next by the observed agent is derived (called *pending set* in PHATT) and thereby constraining the model of observation (observations are mapped to effects of the pending actions to infer the probability distribution for the observed action). Using a hierarchical task network (HTN) representation, an HTN plan conveys both the goal (root of the HTN tree) and the plan (the HTN itself).

HICOR extends PHATT's approach by introducing the world state and metric-time constraints in the modelling of plan libraries. In addition, to determine the prior probability of goals adopted by the observed agent, HICOR takes into account how the current world state affects preferences of the observed agent. This is done by keeping an influence map recording how the different game state features (including units and their positions on the map) affect the desirability of the current state from the opponent's plan standpoint. We also use influence maps to instantiate the plan library by taking into account strategies and tactics that are most consistent with the observations. A further improvement brought by HICOR is the modelling of actions of the observing agent designed to provoke reactions from the observed (opponent) agent – reactions which provide cues as to his intent.

HICOR's Plan Recognition Features

We separate the adversary's plans into strategic and tactical plans. Strategic plans dictate what kind of units the player will produce, whether he will play aggressively, or defensively. Tactical plans dictate how units are deployed and used. These two types of plans are not independent; the strategy a player chooses will affect the composition of his army and therefore will affect how he will use it on the tactical level. It is useful to separate them, however, because the cues used to recognize strategic plans are for the most part different from the ones used to recognize tactical plans. Recognizing these plans separately does not prevent using the output of one plan recognizer as input to the other.

Strategy-Plan Recognition

At the strategic level, the key to victory lies in building an army that makes the most out of each unit's strengths while taking advantage of the opposing army's weaknesses. For instance, while one side may choose to build an aerial army to use the great speed and manoeuvrability of its flying units, this strategy, if discovered early, could very well be countered by the opposing side building up an anti-aerial defence, hence the necessity to predict the opponent's strategy as soon as possible.

Consider a StarCraft duel between a player using the Terran faction and one using the Zerg faction, seen from the Terran player's point of view. Let us assume both players know their opponent's chosen faction, as would be typically the case in StarCraft. The Zerg player's strategy might be to attack very early, in which case the Terran player would have to sacrifice his early resource gathering to build a defence. An alternative strategy may be to gather more resources in the early game to mount a stronger attack later on.

To correctly assess the Zerg's strategy, the Terran player needs knowledge about the typical strategies a Zerg player might use when facing a Terran player – that is, a strategic plan library. Figure 2 illustrates the 12 Hatch strategy using a Hierarchical Task Network (HTN). Note that in StarCraft, early strategies are often named after the first structure(s) built preceded by the number of workers required before that structure is built. For instance, 5 Pool means that the player builds a Spawning Pool after he recruited his fifth worker unit. High level goals (understood as tasks in an HTN representation) are represented with ovals while actions (understood as primitive tasks) are represented with boxes. The actions use the following convention: the first letter is the action taken by the opponent (B for "Build Structure", R for "Recruit Unit" and U for "Research Upgrade") and between parentheses is the object that is being built/recruited/researched (for structures: H stands for "Hatchery", SP for "Spawning Pool", L for "Lair" and E for "Extractor"; for units: W stands for "Worker", O for "Overlord" and Z for "Zergling"; and for upgrades MB is for "Metabolic Boost", an upgrade that augments the speed of Zerglings). If there is a number in the parentheses, it refers to the number of units that should be recruited. The arrows between tasks show precedence relations. The dotted arrows mean that the task needs to be started for the next one to start while the full arrows mean that the task needs to be successful for the next one to start.

For the sake of simplicity, we will stick to opening tasks for this scenario. The opening tasks for the other three strategies are described in Figure 3. To keep the figure simple, some high-level tasks were not expanded. 5 Pool is a very aggressive opening strategy while 12 Hatch is the most long-term oriented strategy. 9 Pool and 12 Pool are middle grounds, with 9 Pool being the more aggressive of the two.

Let us say that, at the third minute of the game, the Terran player notices that the opponent does not have a spawning pool yet, but did not have the time to see the number of workers before his scout was chased away. With the current representation, we do not have enough information to rule out any plan. However, 5 Pool only works because it

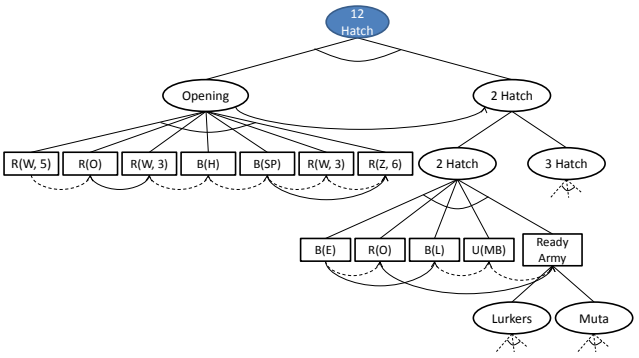

Figure 2: A partial view of the 12 Hatch strategy

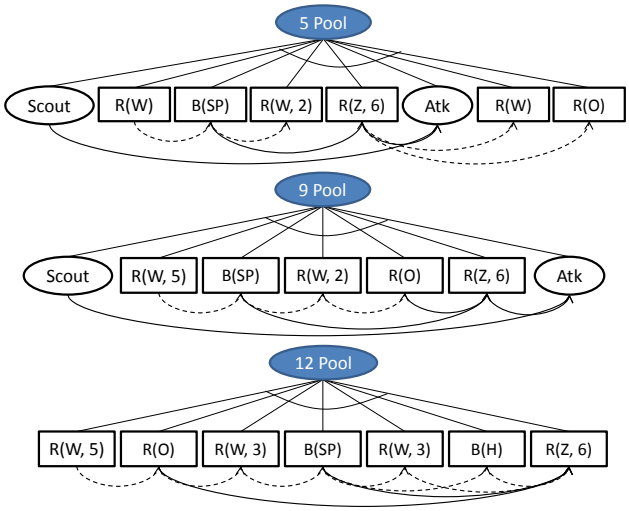

Figure 3: The opening task for the 5 Pool, 9 Pool and 12 Pool strategies

sends a small force very early. We could therefore add to the strategy description that the player must have started his spawning pool before a minute from the start of the game (or the Spawning Pool must be finished by a minute and a half after the start of the game). With this information, the Terran player could rule out the 5 Pool strategy even without knowing the number of workers the opponent possesses.

Tactic-Plan Recognition

At the tactical level, units can be used in multiple ways or can target different units or areas, so it is important for a player to predict how the enemy will use its units to turn the combat in his favour. For instance, if the player can predict that the opponent will try to destroy one of his valuable units, he can move the unit out of range to protect it or even use it to set a trap. Inversely, the opponent would do well to recognize the trap before it is too late.

Let us keep our previous match between Terran and Zerg. The Zerg went with a 12 Hatch strategy and the Terran, after correctly recognizing it, decides to play it safe and gather a maximum of resources to build a strong foundation for a mid-to-late game army. Now the game has been going on for a while and although there have been a few skirmishes, no battle has swung the tide either way. Both players control several important locations, some more critical than others, some better defended than others. A Terran scout suddenly dies after noticing a large army gathering. This army is now on the move and although the Terran player can muster a force strong enough to stop it, he does not have enough time to intercept it before it reaches one of his locations. He needs to send his own army to defend the location the enemy will attack, but with his scout dead, he cannot see where the enemy is heading. Sending the army to the wrong place will most likely cause the Terran player to lose valuable structures and will result in a major set back.

A variety of evidence helps predict where the opponent will attack. His previous actions are some of them. For instance, Where and when did he scout? Did the Zerg scout all of the Terran's locations? Did he attack any locations previously? If so, how successful was the attack? His strategy is also strong evidence, as it affects his army's strengths and weaknesses and also affects the kind of resources or locations he needs to further his strategy. The current situation is another thing that will affect his decision. If the Zerg player holds fewer locations and produces less resources, then he will probably choose to attack a lightly defended Terran position in order to get on par and set himself for a stronger attack later. If, however, the Zerg player has the resource and location advantage, he may opt for a direct assault against the Terran's main base to cripple his enemy and win outright.

Assume the Terran successfully predicted where the Zerg will attack. The two armies meet on either side of three bridges leading to the Terran base, as seen in Figure 4. The armies are equally matched and the combat could easily swing either way. Predicting how the Zerg player will use his units is key to defeating the attackers and protecting the Terran's base.

Figure 4: Terran (bottom right) and Zerg (top left) armies facing off

Understanding Structural Relationships from Observations

HICOR uses influence maps to model the game configuration and the area of influence of all observed units. Influence maps are usually used to plan the AI's course of action in RTS games. Here, we use them to find the most profitable course of action for our opponent and therefore predict his actions. In Figure 4 for instance, the influence map would tell us, amongst other things, that the Terrans have little anti-air influence in the area to the left of the tanks, which would therefore be a good place to attack the tank with aerial units such as Mutalisks. In other words, although there may be many possible plans available to the Zerg player in the current situation, the influence map tells us that plans where the Mutalisks attack from the left are more probable. If the Terran player uses that information to move his marines to the left to protect his tanks, shifting his influence, the probabilities of each possible plans will shift accordingly.

Provoking to Observe the Reaction

As part of the plan recognition process, one could provoke the observed agent to observe its reaction and use it as a cue. For instance, suppose that the Terran player gets the upper hand in the previous fight that followed the encounter of Figure 4 and the Zerg player decides to retreat his surviving units. It is quite possible that he is simply trying to save those units in order to raise another army or help defend against a possible counter attack. In that case, it would be beneficial for the Terran player to pursue and kill as many fleeing units as possible. However, the Zerg player may also be trying to lure the Terran army towards an ambush. In that case, his plan would be to retreat his units, join with reinforcements and wait for the Terran army. However, it is unlikely that the Terran player will get to see those actions (after all, traps only work if the victim is unaware). What he can do, if he suspects an ambush, is to send a single unit rather than his entire army. If the Zerg player set up a trap, he will attack that unit and the trap will be uncovered.

HICOR allows for the modelling of provoking actions by including goals and actions executed by the observing agent into the plan structure. The interpretation of such a plan tree is that the observing agent simply waits for the observed actions to occur, but can decide to execute his own actions whenever they become available. To illustrate, in the HTN plan of Figure 5, the parallelograms are parts of the plan where the observer gets to act in order to provoke a reaction from the observed agent.

Making Active Observations

Active observations concern observations initiated by the intent recognition process. Usually, in RTSs, a player only sees regions around units he controls. The intent recognition process may need information about regions under enemy control to see what the opponent is preparing. In order to get this information, the AI will need to send a scout in that region. This gives an active role to an otherwise passive module and the question now becomes, how much control should the plan recognition process have on scouting. One option could be be to have the plan recognition process take control of a unit and scout as it wants, giving it full control of the information gathering process. Another option would be to have a separate scouting manager that would accept requests from other modules, including the plan recognition algorithm. Since HICOR allows plan libraries including actions of the observing agent, active observation actions just become particular cases of the actions for the observing agent.

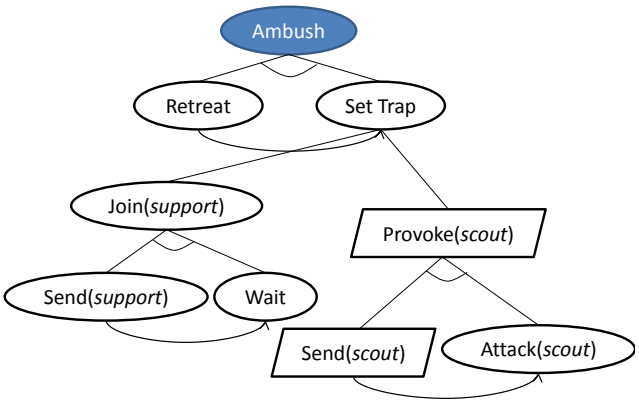

Figure 5: A plan for an ambush with an opportunity for provocation to observe a possible reaction

HICOR's Intent Recognition Algorithm

HICOR's intent recognition algorithm proceeds by updating upon each new observation a current set of PESs, that is, a set of current plan execution statuses, explaining the observations to date. To allow for such reasoning, however, we need to have prior knowledge of the plans the observed agent could be following. Therefore, HICOR uses a plan library to model all the plans our opponent could follow. Whenever HICOR observes an action, it uses the plan library as well as its previous hypotheses to update the possible opponent's PESs. A new observation may contradict previous hypotheses, in which case they are removed, may generate new hypotheses or may support previous hypotheses. This process is shown in Figure 6 (to keep the figure simple, only one plan is shown at a time, but in reality, each PES may contain multiple concurrent plans). The goal of this process is to have the considered hypotheses get increasingly closer to the opponent's actual PES until we recognize it with high enough certainty.

For added flexibility, HICOR uses parameterized plans in its plan library, that is, a plan template using variables as placeholders for quantities (objects, positions, etc.) to be bound dynamically during plan recognition. For instance, the arguments in the plan shown in Figure 5 are parameters instantiated to specific corresponding units, consistent with preconditions attached to plan nodes.

Definition 1. *A plan library PL is a tuple (Π, Pr, O) where, Π is a set of plans, $Pr : \Pi \rightarrow [0, 1]$ is a function that gives the* a priori *probability that the observed agent will have that plan, and O is a set of possible observations.*

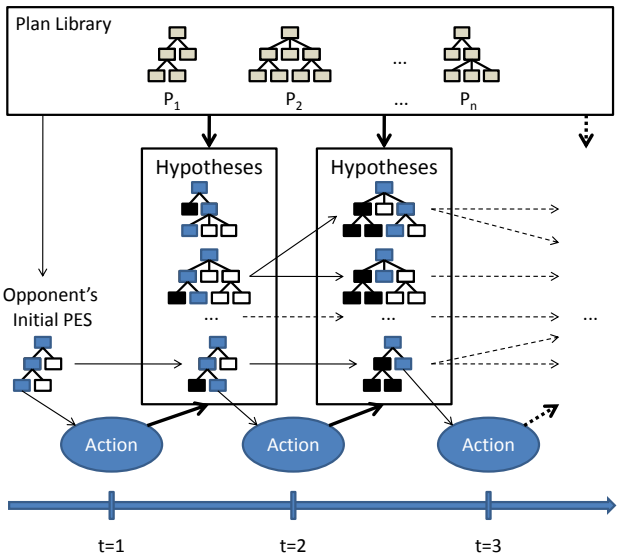

Figure 6: The plan recognition process in HICOR

The above definition does not constrain the formalism used to specify the plans. For HICOR, we chose to specify them in the form of HTNs where precedence constraints can be augmented with time constraints. For instance, in the 5 Pool plan presented in Figure 3, the arrow between $B(SP)$ and $R(Z,6)$ specifies that the $B(SP)$ task must be finished before $R(Z,6)$, but we could augment this arrow with a constraint stating that $R(Z,6)$ must start withing five seconds of $B(SP)$ completion.

Definition 2. *A timed precedence constraint is an expression of the form $u \prec_{tc} v$ where u and v are tasks and tc is a time constraint formed according to the grammar:*

$$tc \rightarrow t < c | t > c | tc \wedge tc$$

where t is the time since u was completed and $c \in \mathbb{R}^+$ is a value in seconds. $u \prec_{tc} v$ means that v must be executed after u, within the time frame specified by tc.

For example, the constraint described above would be written $B(SP) \prec_{t<5} R(Z,6)$.

When a game starts, if we specified multiple plan libraries, we must tell the algorithm which plan library it should use for the game. This allows us to specify a different plan library for each possible match-up and type of games. During the game, the algorithm is given as input a stream of observations, or evidences. The algorithm uses this evidence to generate the set of possible hypotheses H.

Definition 3. *A plan execution status PES is a tuple $(T, \langle En_0, \ldots, En_n \rangle)$ where, T is a forest of partially completed plan trees (i.e., trees with zero or more actions marked as "completed"[1]), and $\langle En_0, \ldots, En_n \rangle$ is the sequence of set of enabled actions generated at each new observation.*

[1]In practice, we do not store entire trees; we only need to store the PES (minus the world state).

When the algorithm starts, the only explanation in the set E is $(\varnothing, \langle\rangle)$, stating that the opponent has no plan. Whenever the algorithm receives a new evidence e_i, it creates a new set of explanations E' by extending all explanations in E to fit e_i in two ways.

First, it tries to match the evidence e_i to an enabled action in each plan tree $\tau \in T$.

Definition 4. *An action is enabled if:*

- *all constraints on it are met, or*
- *there are no constraints on it and all the constraints on all of its ancestors are met.*

If a match is found, it calculates the new set of enabled actions En_i and adds it to the sequence of set of enabled actions. The algorithm then adds $(T, \langle En_0, \ldots, En_i \rangle)$ to E'. Note that this may generate more than one new explanation per explanation in E. Indeed, if e_i fits multiple enabled actions, then a new explanation is generated for each one of them.

The second way to extend an explanation is by adding a new plan to it. To this end, it loops through all plans $\pi \in \Pi$ to try and match e_i to one of π's leftmost actions.

Definition 5. *An action is leftmost if there is no constraint on it nor on any of its ancestors.*

If a leftmost action of π matches, that action is marked as completed and π is added to T. En_i is computed in the same way, but because we assume that the agent had this new plan since the beginning, we "backpatch" all previous sets of enabled actions by adding to each of them all leftmost actions of π. This new explanation is added to E'.

Before extending an explanation, however, $\forall v \in En_{i-1}$ we check all time constraints of the form $u \prec_{tc} v$. If such a constraint is violated, the explanation is removed and not extended. Similarly, when extending an explanation if an action could not be added to the set of enabled actions because it was violating a time constraint of that form, the explanation is not added to E'. Indeed, if the task under an AND-node was not executed in time, we assume it will never be executable again and therefore the whole plan cannot be fulfilled[2].

Probabilities The probability of each plan execution status PES_k is computed as in Equation 1 :

$$Pr(PES_k, \langle e_1, \ldots, e_k \rangle) = \prod_{\tau}^{T} Pr(\tau) \times \prod_{\substack{o, \\ c=\text{child}(o)}}^{PES_k} Pr(c|o) \times \prod_{t=1}^{k} Pr(e_t | En(t)) \quad (1)$$

[2]When a plan contains an OR-node, the algorithm generates a different explanation for each possible alternative. Therefore, if an action under an OR-node violates a time constraint, the whole explanation for that alternative will be removed, but there might still be explanations left for the plan containing that OR-node if other alternatives are still possible.

The first factor of the equation corresponds to the *a priori* probability of each plan tree in PES_k. This function is given with the specification of the plan library.

The second factor denotes the probability of choice in the plan tree. Recall that plans containing OR-nodes generate a different explanation for each possible alternative. The probability of choosing a particular child sub-tree $c \in PES_k$ given its parent OR-node $o \in PES_k$ must thus be considered.

The last factor of the equation is the probability that the evidence e_t (observed at time-step t) is the one chosen next for execution by the agent. This enables the algorithm to consider the sequence of evidence $\langle e_1, \ldots, e_k \rangle$ as an *ordered* trace of execution.

Using Equation 1, the algorithm is able to compute the probability of a goal, given the sequence of evidence, as seen in Equation 2. That is, the algorithm computes the sum of the probabilities of all PES_k in the set of hypotheses H that contain the goal G_i, and divides it by the sum of *all* $PES_k \in H$.

$$Pr(G_i|\langle e_1, \ldots, e_k \rangle) = \frac{\sum_{PES_k}^{H_{G_i}} Pr(PES_k, \langle e_1, \ldots, e_k \rangle)}{\sum_{PES_k}^{H} Pr(PES_k, \langle e_1, \ldots, e_k \rangle)} \quad (2)$$

Influence Map An influence map is a grid overlaid on top of the physical map where each cell has a value representing the influence each side has in that area. In HICOR, the influence is a function of the units' range and fire power. The influence in a cell is represented by a single integer value. If this value is positive, then the cell is under our control whereas a cell with a negative value can be seen as being under the opponent's control. To calculate the influence map, each unit adds (or subtracts) its fire power to the influence of all cells in its weapon range, then adds gradually less influence to the cells outside of its range to model the fact that it can move to attack that cell if necessary[3]. A highly negative influence in a cell means that this cell is very dangerous whereas a highly positive influence means the cell is very safe (or very dangerous for our opponent). We keep both an air and ground influence map because some units can only attack on the ground while others can only attack in the air. Even units which can attack on both levels may not have the same range and power when attacking on the ground or in the air.

The influence map is very useful for tactical plan recognition as it requires high spacial awareness. It is used in two different situations. First, HICOR uses it to recognize and instantiate actions. If a plan contains an action such as "Attack Flank" for instance, we first need to use the influence map to locate our flanks before we can recognize an attack on it. Similarly, if a plan has a high level task "Harass(base)" where base is a variable and "Harass(base)" has a precondition stating that the base must be lightly defended, then we need to check the influence map to find which of our bases are lightly defended before we can instantiate this task for each of those bases.

The second use for the influence map is to update the probabilities for plans or for branches of an OR-node. For example, if we are considering two possible plans, "Attack Flank" and "Attack Front", we can use the influence map to see which one is more likely. If the opponent is using aerial units to fulfil his plan and our anti-air influence is bigger on our front than on our flank, then we can raise the probabilities of the latter plan and lower the probabilities of the former plan, even if *a priori* probabilities tell us that in the abstract, "Attack Front" is more likely.

Preliminary Experiments

HICOR is implemented in Java and is connected with the StarCraft C++ API provided for the AIIDE competition. As mentioned in the introduction, our motivation is to participate in the AIIDE StarCraft competition. This competition comprises of 4 different tournaments, one of which gives the AI complete observation of the opponent's actions. The experiments reported herein are preliminary and consider complete observation of the opponent's actions. Note that even with complete observation, inferring goals remains uncertain.

All the experiments were run on an Intel Core 2 Duo 2.4 GHz. The experiments deal with recognizing the plan of a Zerg player playing against a Terran player. Because we are strictly interested in recognizing adversarial plans and not in planning a response to it, we used replays of Terran vs Zerg games. Our algorithm acts as if it was seeing a game as it is played, from the Terran player's point of view (but again, with full knowledge of the opponent's actions). The replays were taken from the Team Liquid website, one of the biggest StarCraft communities [4]. All games were chosen randomly from a base of competitive Terran vs Zerg games where the Zerg players range from C level iCCup players to top pro players. Those games are completely unbiased as they were played without knowing that they would eventually be used for this experiment. The games' durations range from five to forty five minutes. In these experimentations, we limit ourselves to recognizing strategic plans.

At the strategic level, goals (not plans) are for the most part mutually exclusive. As such, the results below hold for cases where only one goal is considered at a time, even though the algorithm implementation inherently allows multiple goal recognition. We used five different plans similar to the one seen in Figure 2. Two plans have around twenty primitive actions while the other three have around sixty primitive actions. Two have no OR-nodes, one has one OR-node and the other two have three OR-nodes. This may seem small, but it's more than enough to model the typical Zerg strategies against Terran players.

[3]The rate at which the influence reduces depends on the mobility of the unit. It will reduce slower for fast units whereas immobile units will not spread influence outside their range at all.

[4]http://www.teamliquid.net

Currently, HICOR refreshes the plan recognition results every time it gets a new observation. In our experiments, the rate of observation ranges around an observation every three seconds, but this is because we only send HICOR observations related to strategic plans and because there is very little happening in the first minute or two of a game. In theory, however, HICOR with single-goal recognition could refresh at a much faster rate considering that our experiments show that updating the explanations from an observation never takes more than 50 ms. This makes this solution perfect for our real-time needs.

Because it is often impossible to tell some plans apart early on, HICOR relies heavily on the *a priori* probabilities of the plans to tell which one is more likely. If we take the plan with the highest probability as HICOR's prediction, then early on HICOR will be "wrong" very often. This is not how HICOR's output should be used, however. If the plan with the highest probability only has a probability of 25%, it would be silly to pretend HICOR made a wrong prediction when the opponent is actually using another plan. A better way to use HICOR's output is to use a threshold. In our experiments, when we used a threshold of 50%, HICOR made wrong predictions only about 11% of the time. That is, during the whole time where an hypothesis' probability goes above 50%, until the time where the hypothesis is confirmed, the hypothesis given by HICOR is correct for about 89% of that time. HICOR always converges on the correct plan before the plan is finished, including which branch of the OR-nodes will be used.

Those are preliminary results, however. We have assumed full observability of the opponent's action, which makes it impossible for him to hide key elements of his plan from us. Furthermore, because a player almost always has a single strategic plan at a time, HICOR does not have to deal with concurrent plans. Finally, the small number of plans makes it less likely for HICOR to be wrong. When we will move to experiments with tactical plan recognition, we expect this to take more CPU time [5] and lead to more incorrect predictions. This will most certainly lead us to examining improvements to the current implementation.

Conclusion

In this paper we discussed some of the the main challenges in building an RTS plan recognizer. We also introduced a new solution approach to the problem along with results from a preliminary implementation. The preliminary results are promising. The addition of metric time already simplifies significantly the plan recognition problem in RTSs, allowing for the creation of both more descriptive plans as well as more generic ones. It allows for plans that would otherwise be very difficult, or even impossible to specify.

The continuing implementation will focus on testing with more complex plan libraries. This includes developing tactical plan libraries and scenarios with multiple concurrent goals. In the current experiments, we have not yet dealt with active observations and provoking actions. We have also limited the scenarios to strategic plan recognition, assuming one strategy goal being pursued at a time. All these features provide additional dimensions for validating the current implementation. Integration with a decision making module is another area of current investigation. This will lead to a fully-fledged RTS AI, making use of recognized intent, capability and opportunity of the opposing force to decide and execute plans against the force. All these incremental developments will converge towards the ultimate objective of participating to the announced AIIDE StarCraft competitions.

[5] Multiple goal recognition leads to a time complexity exponential in the size of the conjunctive goal because this lead to an exponential explosion of possible PESs.

Acknowledgement

The work presented herein was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada. We would also like to thank Robert Goldman for numerous exchanges helping us understanding PHATT.

References

Avrahami-Zilberbrand, D., and Kaminka, G. 2007. Towards dynamic tracking of multi-agents teams: An initial report. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*, 17–22.

Blaylock, N., and Allen, J. 2006. Fast hierarchical goal schema recognition. In *American Association for Artificial Intelligence (AAAI)*.

Bui, H.; Venkatesh, S.; and West, G. 2002. Policy recognition in the abstract hidden markov model. *JAIR* 17:451–499.

Geib, C., and Goldman, R. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 117(11):1101–1132.

Goldman, R.; Geib, C.; and Miller, C. 1999. A new model of plan recognition. In *fifteenth Uncertainty in Artificial Intelligence Conference*.

Kautz, H. A. 1991. A formal theory of plan recognition and its implementation. In *Reasoning About Plans*. Maurgan Kaufmann Publishers. 69–125.

Kott, A., and McEneaney, W. M. 2007. *Adversarial Reasoning: Computational Approaches to Reading the Opponent's Mind*. Chapman & Hall/CRC.

Pynadath, D. V., and Wellman, M. P. 1995. Accounting for context in plan recognition, with application to traffic monitoring. In *Proc. the Eleventh Conference on Uncertainty in Artificial Intelligence*, 472–481.

Rabin, S. 2005. *Introduction To Game Development*. Rockland, MA, USA: Charles River Media, Inc.

Schmidt, C. F.; Sridharan, N. S.; and Goodson, J. L. 1978. The plan recognition problem : an intersection of psychology and artificial intelligence. *Artificial Intelligence* 11:45–83.

Schmidt, C. F. 1976. Understanding human action: Recognizing the plans and motives of other persons. *Cognition and Social Behavior*.