Video Game Strategy Inference Given Partial Observability using Discrete Probability Densities

Drexel University – College of Computing and Informatics
Computer Science Master's Thesis for Michael Kozak
mike.w.kozak@gmail.com
Thesis Advisor: Santiago Ontañon
so367@drexel.edu

Abstract

This paper presents an examination of the application of probability density functions to strategy identification in partially observable game spaces. Specifically, we focus on the design, implementation, and evaluation of the Intent Recognition Engine (IRE) for NOVA, an autonomous agent designed to participate in the StarCraft AI competition. StarCraft is a video game in the "real time strategy" subgenre that pits humans and AI against each other in real-time combat simulations where each player controls a virtual economy and an army and uses these to defeat the opponent. We describe the challenge of determining enemy strategy with only a partial view of their total forces, of mapping strategies to a coordinate space, and of performing localization in that space. In addition, we present the results from evaluation of both win rates against enemy AI as well as prediction accuracy given post-game ground truth. Finally, we conclude with a discussion of remaining challenges and opportunities for further development.

Index Terms – Game AI, Real-Time Strategy, StarCraft, Partial Observability, Probability Density Function

I. Introduction

Early prediction of enemy strategies and tactics produce a marked improvement in effectiveness against intelligent opponents (Laviers, 2009). Real-time strategy (RTS) games have become a rich platform for the development and advancement of related AI techniques due to their decision complexity, partial observability, and their variable determinism (Santiago Ontañon, 2013, 5 (4)). This research has led to the production of many bots with varying strategies which compete in yearly AI competitions such as the "AIIDE StarCraft AI Competition" and the "CIG StarCraft RTS AI Competition". NOVA (Alberto Uriarte, 2014) is one such bot first developed in 2013 at Drexel University. This thesis focuses on the development and evaluation of The Intent Recognition Engine (IRE), an inference system that applies probability density functions to strategy identification in partially observable game spaces. IRE builds on NOVA capabilities with the goal of demonstrating its predictive accuracy against enemy bots.

RTS games present several open challenges for traditional decision-making systems due to the large number of possible moves that every unit can make when combined with the large number of units that can exist at the same time. Combined with the partial observability of the environment, this results in a state space that is far, far too large to search using traditional methods. As a point of comparison, Chess has a state space of 10^{47} and the board game "Go" has a state space of 10^{171}. StarCraft, on the other hand, has a state space of over 10^{2791} (Uriarte, 2017). In addition, the real-time nature of the genre means algorithms must act quickly, effectively ruling out many classes of optimal decision-making algorithms. Developing techniques that can rely only on observable data to make inferences about higher level enemy decision making, without having to explore the full potential state space, can help ameliorate this problem by accepting sub-optimality in exchange for efficiency. Effective implementation of these techniques can produce inference systems that are effective across many types of partially observable environments.

This paper aims to detail one such technique that applies algorithms developed for geolocation given imperfect information to identification of enemy strategies by mapping them into an N-dimensional space. It is organized as follows: Section II introduces the RTS game StarCraft and its unique challenges, the concept of mapping strategies to points in space, and probability density functions. Section III reviews the algorithmic details of IRE detailing how observations in-game are converted to strategy predictions and, ultimately, counter-strategies. Section IV details the test procedure and evaluation results of IRE trials against other competitive bots. Section V touches on related work being performed in this area. Section VI outlines the conclusion made from this research and development effort. Finally, Section VII outlines future work to be performed on the algorithm to further improve results.

II. Background

RTS Games

Real-Time Strategy (RTS) games are a sub-genre of grand strategy games where players are tasked with gathering resources to construct buildings, train units, and research upgrades in order to achieve victory over one or more opponents. Although some games offer a variety of "win conditions", the vast majority define victory as the total destruction of enemy assets. RTS games have become a hotbed of AI research and the next step in the evolution of adversarial AI beyond Chess and Go, in part because they offer the following unique challenges:

- Unlike Chess and Go, both players can perform actions simultaneously. In addition, while some moves are limited only by the speed at which the player can perform them (further limited by the rendering frame rate), other moves are *durative*, meaning the moves themselves take time to complete and in some cases can be interrupted by the player or the enemy.
- By nature of allowing simultaneous actions, RTS games are real-time, meaning game state advances regardless of player action. All other things being equal, the player which can act faster has an advantage over the other players. This means that fast, effective decision making is required for competitive systems.

- Whereas with Chess and Go the entire board is visible at all times, many RTS games have partial observability through a "fog of war" (Figure 1) that limits view into enemy activity. All game state changes that take place outside the combined visibility ranges of all friendly assets are invisible to the player until that area is observed.
- Many RTS actions are non-deterministic, meaning they have a probabilistic chance of success or failure and thus the results cannot be predicted with perfect accuracy.
- The combined complexity of all possible unit actions across all existing units against all known, previously known, and unknown enemy states significantly exceeds that of Chess and even Go in terms of computational complexity. Where Chess and Go are EXPTIME-complete AI problems, StarCraft is at best 2-EXPTIME (Uriarte, 2017).

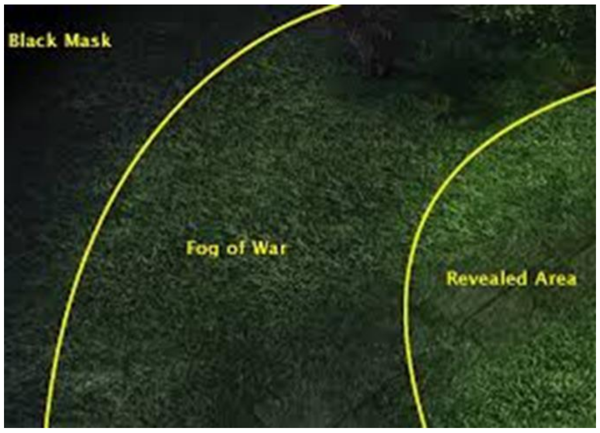

Figure 1- In most RTS titles, you can only observe what appears in the revealed area of your units sight ranges. Objects in the fog of war will not be updated, and objects in the black mask will not be known until first revealed.

StarCraft

Figure 2 - StarCraft games involve heterogeneous armies battling until only one remains

StarCraft: Brood War (Figure 2) is a RTS game released by Blizzard Entertainment in 1998. Since then it has become a cultural and competitive phenomenon, going as far as to become a televised sport in some countries. The game is set in a science-fiction universe consisting of three races: Terran, Protoss, and Zerg. Each race consists of unique buildings, units, and abilities with a heterogeneous set of capabilities between the three races. Each race specializes in different combat styles: balance for Terran, power for Protoss, and speed for Zerg, making the right strategic decision a factor not only of current resources but of enemy race. That also means that enemy strategies will vary from race to race and an effective counter-strategy for one race will not apply evenly to all races. In addition, and after over a decade of regular software updates, it has also become one of the most balanced games in the genre, making it an ideal choice for the development and evaluation of AI systems against peer adversaries with heterogeneous capabilities.

Strategy Representation

In order to identify opponent strategies, it is first necessary to produce a representation of strategy that is robust to the various tactics across all three races. Let $R = \{T, P, Z\}$ be the set of all races. Further, let n be an asset (a unit or building). In StarCraft every unit, building, and research upgrade has a fixed set of pre-requisites that must exist before the object can be produced. Each race also has one "command center" building, hereafter denoted n_0, which has no pre-requisites and is the only one capable of producing worker units that are required to gather resources and build additional buildings. Let $N = N_T \cup N_P \cup N_Z$ be the set of all unit types in the game, defined as the union of the unit types of each of the three races (where the sets of unit types for each race are disjoint). When we combine all these components for a given race, it produces a single "tech tree" TT representing the races' complete set of capabilities (Figure 3). The tech tree for a race r, TT_r, is a tree with a root node n_0 and one node per unit type in N_r, and where the edges define dependency relationships among them.

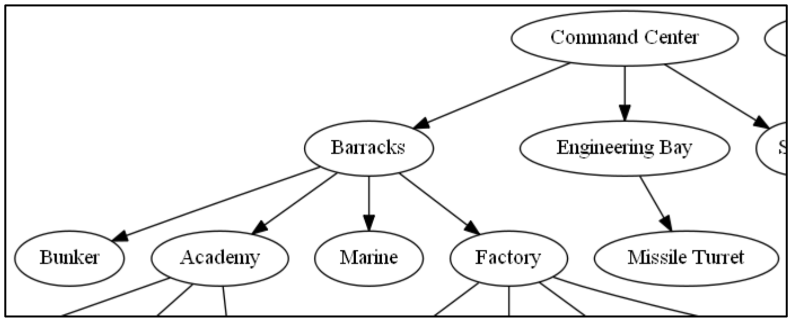

Figure 3 - A segment of the Terran Tech Tree showing some of the dependencies between units and buildings, which are always one-directional.

Stemming from this representation is the notion of "build order". A build order is a unique set of ordered productions (building, unit, or upgrade) designed to produce certain units in certain quantities necessary to execute a specific strategy. This build order induces a subtree of the races' tech tree, consisting of only the nodes and edges required to achieve that series of productions. We define a strategy s for race r as a subtree of TT_r. Strategies in StarCraft can generally be represented by a build order, and while there is nuance to the exact execution of that build order, such as building location placement, certain aspects remain fixed due to the strict dependencies between objects, such as requiring a specific building type to produce a specific unit type.

Strategies can vary heavily depending on the types and quantity of units generated. Some strategies rely on aggressive assaults while others are designed to defend until a large enough army can be produced. Similarly, while some strategies are more effective against ground units, others are more effective against air units. IRE implements a classification method that captures these differences in a way that aims to minimize or eliminate duplicate classifications. It attempts to perform strategy inference in part by characterizing strategies using a fixed number of parameters. By selecting a single set of quantifiable parameters, we can give every strategy a unique "fingerprint" consisting of a measurement for that strategy against each. IRE treats this complete set of numerical values for each strategy as a set of coordinate intensities, not unlike a vector in a Cartesian space. We represent this coordinate vector as a value along each axis, with a range of -1 to 1, where each axis is one of the aforementioned quantifiable parameters. Together, this N-dimensional space contains all possible unique combinations of strategy parameters, henceforth denoted S or the “Strategy Space”. A point in Strategy Space, then, represents a possible strategy whose parameter intensities match the location provided by the point.

IRE uses a 3-axis system to represent the Strategy Space for StarCraft. The first axis measures a strategy’s focus on ground units versus anti-ground units. The second axis measures a strategy’s focus on air units versus anti-air units, and the third axis measure’s a strategy’s overall aggressiveness versus defensiveness. The closer to zero a point is along any given axis, the more “balanced” that strategy is in producing units that can achieve, or fails to achieve, both extremes. For example, a strategy that emphasizes aggressively building ground units that cannot attack air units to attack without concern for self-defense would measure as a 1 on the ground axis, a 0 on the air axis, and a 1 on the aggression axis. Conversely, a strategy that involves building heavy base defenses to produce a more advanced aerial armada would measure as a -1 on the ground axis, a 1 on the air axis, and a -1 on the aggression axis.

We selected these axes based on the fixed nature of unit capabilities in RTS games. All units in StarCraft are either air or ground units, and each can either attack ground units, air units, or both. In addition, successful strategies in StarCraft either emphasize attacking as quickly as possible (“rushing”) or defending against opponent “rushes” in the hopes of producing more powerful units for a counter-attack. These axes were evaluated against a set of common build orders modeled by IRE to ensure that no two strategies had the same fingerprint. Although these particular axes were chosen due to their relevance to the most common build orders, any arbitrary axis could be selected and added to the code without consequence to IRE. For testing, we codified and then scored along each axis the 18 most common build orders as identified using clustering and quantitative analysis of the 2011 and 2012 AAIDE StarCraft AI competitions (Santiago Ontañon, 2013, 5 (4)), referenced in the tables below.

Terran Strategy	**A vs AA**	**G vs AG**	**Atk vs Def**	**Description**
Bio	-0.25	0.75	1	Marines and Medics buildup and then attack
Rax_fe	0	0	-1	Build a second base to amass as many marines as possible
Two_facto	0	1	0.5	Build 2 factories to produce tanks
Vultures	0	1	0.75	Rush build Vultures
Air(wraiths)	0.75	-0.5	0.25	Defensively play until you get wraiths for assault on enemy resource gatherers
drop	0.25	0.25	-0.25	Bio strategy but with airdrops for moving large numbers of units behind enemy lines
Protoss Strategy	**A vs AA**	**G vs AG**	**Atk vs Def**	**Description**

Two_gates	0	1	1	Build a second gateway to mass produce zealots
fast_dt	-0.25	0	0	Rush to build dark templars
Templar	-0.5	0	0.5	Rush to build and upgrade templars
Speedzeal	0	1	0.75	Rush to build zealots and their upgrades
Corsair	-1	0	-0.75	Air rush
Nony	0	0	0.5	Mass dragoons for tank rush
Reaver_drop	0.25	0	-0.25	Airdrops using reavers
Zerg Strategy	**A vs AA**	**G vs AG**	**Atk vs Def**	**Description**
Speedlings	0	1	1	Zergling rush
Fast_mutas	1	-1	0.25	Mutalistk rush
Mutas	0.75	-0.75	-0.25	Mutalisks with upgrades
Lurkers	0	0	-0.25	Lurker Rush
Hydras	-0.25	1	0.75	Hydralisk Rush

At the start of the game, IRE must first create an instance of the 3-axis Strategy Space and populate it with all 18 strategies represented in the tables above. IRE represents each strategy as a unique subtree of the overall tech tree for the given race with a shared root of n_0. The result is that the Strategy Space for a given race consists of a single node at the origin (0,0,0), with edges from that root representing the specific subtrees for different strategies (Figure 4).

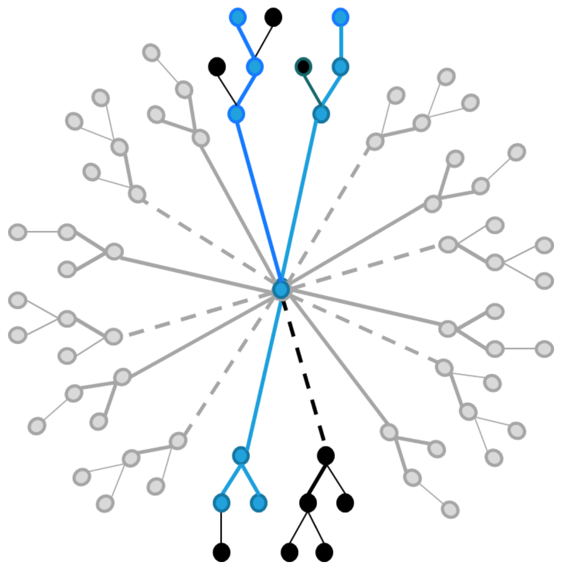
Figure 4 -The Strategy Space for a race consists of a root node at the origin with different strategies branching out in many different directions

To be placed into Strategy Space, each node in the subtree must be given a coordinate in the coordinate frame. Not every node will have the same "position" in space although some may share coordinates within the same strategy. The node position for a given strategy depends on the importance of that node to the strategy. For example, a strategy revolving around the Terran "Vulture" unit relies heavily on those units to succeed, whereas the "Factory" building is only a prerequisite to producing those units. As many of the buildings you can create at the start of a game are required for most strategies, by nature of being common prerequisites, they will appear in many different strategies. As such, we want to make sure that those “common” buildings and units are closer to the “origin” coordinates of Strategy Space than those that represent the culmination of the strategy.

To achieve this (Figure 5), IRE takes each strategy and determines the maximum depth of its subtree (*node_depth*). It gives the lowest most leaf in the tree (*MAX_DEPTH*), the most advanced unit or building in the strategy, the same position as the “score” for the strategy (*strategy.axis.value*). From there, IRE scales all parent nodes linearly from that position to the origin (*node.axis.position*). So, the lowermost leaf node in a strategy’s subtree for a strategy with an aggressiveness “score” of 1 would have an aggressiveness axis position of 1, whereas a node halfway between that leaf and the root would have an aggressiveness axis position of 0.5. Once every node in the strategy subtree has been given a position in the coordinate space, the strategy subtree is merged with the Strategy Space tree by replacing the strategy root with the Strategy Space root node n_0.

```
onAddStrategy(NEW strategy)
      FORALL nodes in strategy.buildTree
            FOREACH axis in StrategySpace
                  node.axis.position = node_depth / MAX_DEPTH
                                          * strategy.axis.value
      graph_merge(strategySpace, strategy)
```
Figure 5 - Pseudocode for adding a strategy to Strategy Space

Probability Density Functions

The ultimate goal of IRE is to infer, given incomplete information, as to what strategy the enemy is performing so that it can employ an appropriate counter-strategy. In order to predict enemy strategies in partial observability, IRE must take a probabilistic approach since perfect data will almost always be unavailable. This also benefits early prediction as similar strategies may look nearly identical during early phases of gameplay when the enemy is producing units and buildings shared between strategies, but diverge later when advanced units become available. As such, IRE extends the concept of mapping strategies into a coordinate space to “geolocate” the actual enemy strategy in this space. It does this using probabilistic methods, in particular Discrete Probability Distributions (DPD) applied to Probability Density Functions (PDFs).

In probability theory, a probability density function (PDF) is a way of describing the relative distribution of a continuous random variable within a bounded sample space (Robert Grover Brown, 2012). They are primarily used to represent situations where the probability of a specific value for the variable is zero due to its continuous nature so instead samples of regions are used to infer the relative likelihoods of the variable appearing. PDFs have many uses, but one such use is in the localization of objects in space (K. Wendlandt, 2005). Using only a signal intensity and bearing to a transmitter, a receiver can use repeated measurements to reduce the number of possible places the object could be until eventually it converges with very high probability on the exact location (S. Venkateswaran, 2013). To achieve this effect, each individual measurement produces a discrete probability density (DPD) map which represents a PDF for the target from a given source of measurement. These DPD maps are layered on top of each other, and overlapping areas represent higher probability locations for the target. Over time and with additional observations, the actual location of the target will emerge with very little error as the most overlapped location additively rises above all other possibilities. (J. T. Isaacs, 2014)

IRE applies this concept to locate an enemy strategy in its 3-dimensional Strategy Space (Figure 6). At the start of a round and before IRE observes a single enemy unit or building, the probability of the enemy using one particular strategy is equal across all strategies. As the game advances and the enemy starts to construct buildings and train units, certain strategies become more likely. For example, an enemy building a large number of air units isn't likely to be pursuing a ground-based strategy, and so strategies that rely on air units become more likely than strategies that rely on large numbers of ground units. Eventually the observed quantity and type of units and buildings narrows the list of possible enemy strategies down to only a few, giving IRE the ability to accurately identify and counter them.

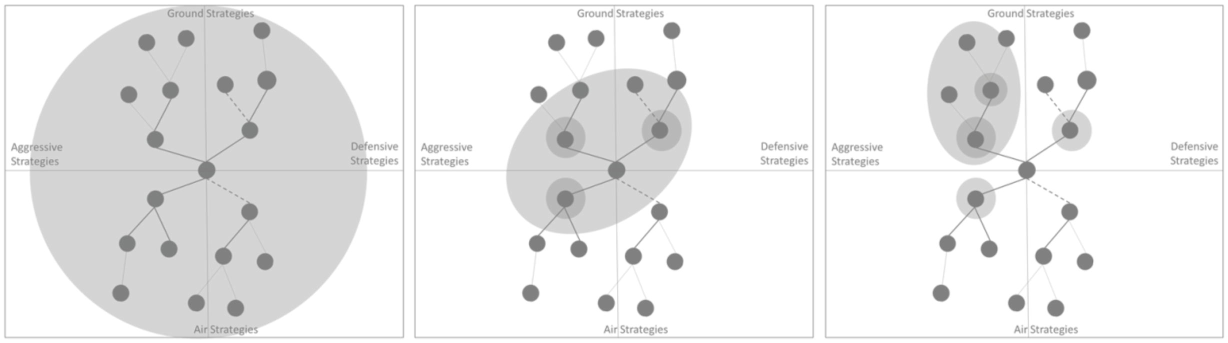

Figure 6 - Initially all strategies are possible. As observations are made, the error bounds around the actual strategy being used converges, eventually settling on the actual strategy being used

To do this, IRE treats an observed enemy unit or building as an "emission" in Strategy Space. A strategy must inherently contain units and buildings, and so each one observed provides an error-bound estimation of the enemy's strategy. Ground units will appear in ground strategies, air units will appear in air strategies, late-game units and protective structures will appear in defensive strategies, etc. A unit or building type may appear in multiple strategies, and as such no single observation is sufficient to confidently predict the enemy strategy. However, even a single observation can provide guidance: a unit that appears in multiple ground strategies will necessarily imply that, if however slight, the enemy is more likely to be attempting a ground strategy versus an air strategy if that is the only unit observed.

III. Converting Observations into Predictions

IRE uses the concepts outlined above in conjunction with observed enemy units in order to make predictions about enemy strategies. After each prediction, IRE uses the results to bias NOVA production of units to ensure a proper balance of effective counter-units to maximize effectiveness in battle. For example, if IRE predicts that the enemy is using an air strategy, it will instruct NOVA to dedicate more resources to building anti-air units. To enable this, we integrated IRE into NOVA which is itself integrated into the BWAPI framework (Figure 7). BWAPI interfaces directly with StarCraft to provide data to custom AI, providing only data a real player would have access to during gameplay to properly simulate real

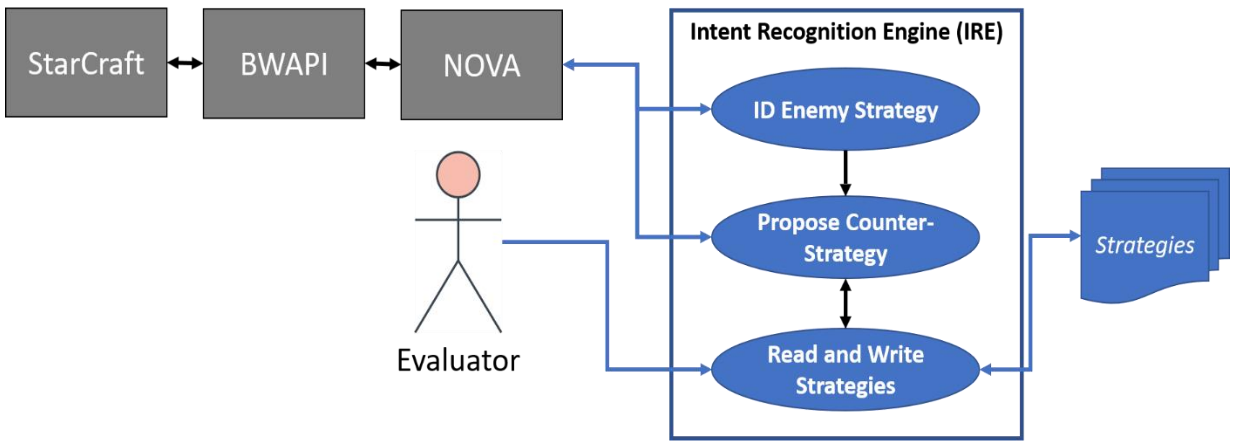

Figure 7 - IRE uses observed enemy units to infer the opponent's possible strategies and use the most likely candidates to bias unit production towards effective counter-units

play. IRE relies on NOVA to command units and manage resources, focusing solely on strategy identification and selection.

BWAPI sends a callback directly to NOVA when a friendly unit observes an enemy unit during gameplay. IRE piggybacks off this callback, but only after NOVA determines that the observed enemy unit is a new unit. When the new unit is detected (Figure 8), IRE goes through all the possible enemy strategies looking for those that also contain the same type of unit. When a match is found, the “intensity” of that node is increased which represents an increase in the likelihood for that node’s strategy over other strategies that do not rely on that unit type.

```
onObserveEnemyAsset(NEW observedAsset)
        FORALL Strategies
                FORALL nodes in Strategy
                        If(node == observedAsset)
                                node.intensity++
```

Figure 8 - Pseudocode for how IRE handles the detection of a new enemy unit

Each observation may increase the intensity of nodes across multiple strategy trees. However, as the game progresses and observations are continually made, the number of potential strategies at play decreases. This is due to the fact that as the game goes on, the enemy will theoretically be making advancements towards completion of one strategy, producing units and buildings specific to that strategy. With more observations, the unique combination of units and buildings being produced becomes less likely to occur across multiple strategies. Eventually one strategy will emerge as the most likely because its nodes will have the highest total intensities relative to the other strategies. Producing units takes time and resources, and so for example IRE makes the assumption that the probability of an enemy attempting an anti-air strategy by producing units incapable of achieving these goals approaches zero as the number of observed units increases.

IRE does not need to wait until one strategy is the clear winner to act, though. The gradually increasing predictive accuracy that comes with a decrease in likely strategies means that IRE and NOVA can begin producing a mix of relevant counter-units as soon as the first observation. As the number of likely strategies decreases, so too will the most effective counter-units. This also means that if the enemy changes strategies, those new observations will eventually converge on a different part of Strategy Space as the node intensities for the new units overcome the intensities of old nodes.

After each observation, IRE makes a prediction as to which strategy the enemy is attempting (Figure 9). First it takes all the nodes across all strategies and sorts them by intensity. The result of this action is that the most observed nodes appear at the top. Although not shown, IRE takes the additional step of removing certain "universally common" units and buildings from the list that are likely to appear at the top of the list by nature of being necessary for all strategies. It than takes the top nodes in the list where the exact number is configurable (*TOP_NODE_COUNT*) and averages their coordinates. Since each axis represents an aspect of strategy, averaging their positions is one means of determining with what

```
onDecideStrategy()
    SORT(strategies.nodes, node.intensity, greaterThan)
    FOREACH axis in strategyAxes
        FOR top TOP_NODE_COUNT nodes in sortedNodes
            totalAxisValue += Node.axis.value
        axisAverage = totalAxisValue / 5
        suggestedCounter = axisAverage * -1
        setBuildPriority(axis, suggestedCounter)
```

Figure 7 - Pseudocode for how IRE converts enemy observations into counter-strategies

intensity the enemy is acting along each axis. For example, if an enemy is building exclusively air units, the expectation is that the average among the most frequently observed nodes would be heavily biased towards "air" on the air versus anti-air axis. If the enemy has a mix of air and anti-air units, the average would converge around the origin. IRE uses the relative intensity along each axis to bias unit production in NOVA which will naturally result in units that would win in combat, such as producing anti-air units to counter an air strategy. This also means that "hybrid" enemy strategies will result in "hybrid" counter-productions, avoiding a potential exploit where the most produced unit is distinctly opposite from the actual strategy being used.

Early in the game, IRE may find itself averaging nodes across both air and ground strategies, which would result in a balanced counter-strategy of units capable of fighting both. As the game progresses, though, and the opponent begins building more specialized units, the resulting strategies should begin to coalesce at one end of the axis, causing IRE to recommend higher production of specific counter-units relative to other unit types. As a result of these actions, IRE ensures that NOVA is always producing units that are best suited to counter not only the observed enemy units, but their overall strategy before said strategy is fully executed.

IV. Experimental Evaluation

Overview

The semi-random nature of enemy AI in videogames makes normal evaluations difficult. A single trial, no matter how thorough, isn't likely to exercise all paths through the system and even a carefully selected series of trials won't account for the randomness inherent in enemy decision making. As such, IRE evaluation is a natural fit for Monte Carlo simulations. To properly determine IRE's effectiveness, we performed many hundreds of semi-randomized trials. These results were collected, evaluated, and are summarized below. Access to source code, documentation, and examples can be found at https://github.com/mikewkozak/IRE for anyone interested in reproducing these results.

Metrics

IRE evaluation focused on three primary metrics: overall win rate, enemy-specific win rate, and prediction accuracy. The first two metrics are primarily concerned with ensuring that IRE improves or at least matches the current effectiveness of NOVA. The final metric does not consider existing NOVA performance. All trials were performed with 2 configurations of NOVA: one with and one without IRE. The evaluative procedures for each metric are as follows:

Metric 1: Win Ratio

Of the three, this metric is the most straightforward. The total number of wins across all trials was divided by the total number of trials performed for both configurations of the system, with the goal being an improvement in win rate when using IRE.

Metric 2: Race-Specific Win Ratio

NOVA does not perform equally well against all enemy races. To further examine the nuances in benefit that IRE provides, both configurations were examined on a per-race basis. The trials performed for each configuration were first sorted by opposing race. Each sorted list is then evaluated using the same win ratio process as above, with the goal being an improvement in the win rate for at least once race when using IRE.

Metric 3: Prediction Accuracy

Perhaps the most important metric in evaluating IRE is the effectiveness of the strategy prediction algorithms. Even if other aspects of the AI are flawed, an accurate predictor not only proves the

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Figure 8 - The Jaccard Index measures the overall predictive accuracy of IRE against enemy strategy selection

concept but could also generalize to another StarCraft or RTS AI. To measure this, we rely on the Jaccard Index of the system (Figure 10), which measures the similarity of sample sets. In this instance, A is the number of accurate predictions made in testing, and B is the total number of predictions made. BWAPI provides the ability to query for the complete list of enemy resources at game completion and this ground truth is used to compare the predicted strategy against the real strategy.

Evaluation Procedure/Setup

StarCraft was run using BWAPI 4.2.0 and NOVA to allow for automated logging and trial restarts. The BWAPI "Chaos Launcher" provided the means of starting the game, automatically initializing a match against a bot, and restarting a new game when the existing one concludes. For standardization, all trials both with and without IRE were performed against the default StarCraft AI with its race set to Random. Every trial was run on the AAIDE approved map "Benzene" with a timeout of one hundred thousand "frames" to prevent issues where neither AI can completely eliminate the other and a stalemate occurs.

For data collect, we first had the NOVA StarCraft AI fight against the default StarCraft AI for 300 consecutive games. We saved those logs to a file, and then we ran NOVA again for the same number of trials but with IRE enabled to make predictions and bias unit production. The logs included not only the game time (in frames), game winner, enemy AI race, and score (as tallied by StarCraft), but also the complete list of enemy units and buildings produced during the game, including those never discovered by NOVA. We used this list to evaluate IRE trials in determining accuracy of enemy strategy predictions, using the ground truth of their actual unit productions as the basis of evaluation. The logs from all runs collected into a single location and then scraped for the required data for performing the metrics evaluations as described above.

Results

Base NOVA averaged a win ratio of 79.92% across all trials. NOVA with IRE providing additional support by comparison had an average win ratio of 79.2%.

NOVA averaged a win rate of 25.97% against Terran, 98.89% against Protoss, and 84.45% against the Zerg. Comparatively, NOVA with IRE averaged a win rate of 4% against Terran, 67% against Protoss, and 84.7% against Zerg (Figure 11).

The overall predictive accuracy of IRE was XYZ%. Surprisingly, a larger number of wins against the enemy AI were extremely early in the game, before they were able to produce any offensive units at all. Because of this, many predictions were erroneously true, in the sense that any possible strategy was still valid at the

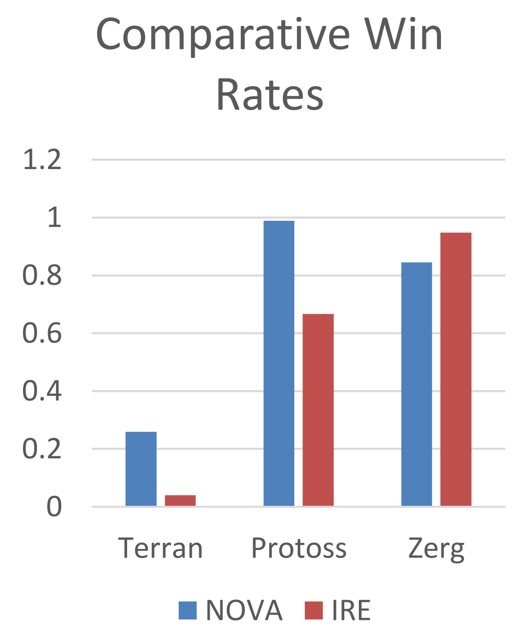

Figure 9- These details the win and loss rates for all opponents fought using NOVA and with NOVA+IRE

time of victory. In addition, while the actual strategy performed in longer games did not always match the top predicted strategy, it was in nearly every instance at least in the top 5 identified strategies, and therefore had significant impact on the identified counter-strategy. It is our belief that losses incurred due to IRE were primarily a byproduct of disrupting the early game flow and losing to an enemy rush while NOVA was in the process of producing the prerequisite buildings to create more advanced units. We believe that a time-horizon based approach to strategy predictions that allows NOVA to act using its default strategy for the first minute or so before suggesting alterations, a key time when defenses can be built against enemy attacks, might improve win rates. However, further testing will be necessary.

V. Related work

The work presented in this document is related to work in the field of planning in partial observability, specifically Opponent Modeling, Plan Recognition, and Markov Decision Processes. Each of these techniques takes a unique approach to modeling and predicting adversarial intent and comes with their own strengths and weaknesses relative to IRE.

Opponent Modeling

Opponent Modeling is likely the most sophisticated approach among comparable approaches in terms of implementation. It uses game theoretic modeling to reason not only on friendly and enemy actions, but the interactions those actions have on affecting each other. Various implementations have been tried in this space, with more recent efforts relying on deep reinforcement learning to automatically learn enemy behavior patterns and exploiting them (He He, 2016). This approach is effective for deciding how to act as one object against another but is not generally applied to higher level reasoning approaches to *strategy* management. Particularly, game theoretic approaches are often slower due to the inherent complexity of modeling the dynamics of interdependent decisions.

Plan Recognition (Expert Systems)

More traditional AI methods have relied on an expert-systems approach that codified subject matter expert knowledge into a set of observable rules which can be reasoned on. This is also effective for declarative, constraint-based reasoners that base conclusions on strict guidelines. While these systems are less flexible than learning methods, they are generally highly effective in a constrained environment where the most effective choices can be fully modeled. The strategies that IRE reasons on, for example, are based on these very same sorts of strict hierarchical rules. However, IRE extends beyond expert systems by reasoning on what *has not yet been observed* as well as what has been seen by looking at the causal connections between units and their parents/dependencies across many possible strategies and accepts that an early conclusion may be technically true given the available data, but ultimately wrong without having to undo observations already made.

Markov Decision Processes (MDTs and POMDPs)

Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems solved via dynamic programming and reinforcement learning (Markov Decision Process, 2018). Partially Observable MPDs (POMDPs) extend this concept by assuming that we cannot directly observe the transitions between states and can only probabilistically guess what they may be. POMDPs have been used for predicting opponent state for many years but suffer from extremely slow performance due to the massive number of possible states in complex scenarios over time.

To overcome this computation hurdle, more recent research into using MDPs for real-time systems has focused on Markov Decision Trees (MDTs). By building these representative trees, a Monte Carlo Tree Search (MCTS) process can be used to find path that results in the most likely win conditions. MCTS is a heuristic-based process which trades optimality for speed and in-turn makes it more effective for the RTS genre. MCTS was used in the implementation of the AI for the grand strategy game Total War: Rome 2 (Champandard, 2014).

VI. Conclusions

Overall, IRE proves there is applicability in using geolocation techniques to perform higher level strategy inference. By categorizing potential enemy tactics across a uniform set of axes, observations about enemy activities act as beacons which are useable by systems to help geolocate the enemy strategic position in this strategy space. Although, more work can be done to boost effectiveness, IRE shows that these concepts produce promising results worth further exploration. Partial observability will continue to be a hot-button issue for autonomy research, both against human and autonomous opponents, and will continue to be the reality of operational environments for many years to come.

VII. Future Work

Due to the limited scope of IRE's development, there is clear room for growth and improvement not only in the existing algorithm but in adding complementary techniques and more nuanced responses. For example, a deep dive into effective counter-strategies could reveal cases where a specific, targeted response is more effective than biasing build priorities. In particular, results against the Terran race imply that IRE can do more in detecting and responding to the enemy strategy against that particular race.

Currently, edges in strategy graphs represent absolute pre-requisites between nodes. In other words, it is not possible to build an asset until its parent has been build. However, certain units and buildings are complementary: strategies may benefit from support units not explicitly part of the strategy. For example, if a player or AI is building a large number of ground soldiers and they are flush with resources, they may choose to build support units that can heal them. As the duration of the game goes on, probabilistic inferences can be made about the likelihood of those units appearing. In other words: given enough observed solders, it is highly likely that healer units also exist in the enemy army even if they haven't been observed. These probabilistic links could be added to each strategy and strengthened whenever one of the two nodes is observed. Eventually, IRE could respond to a sufficiently high probability of those units existing by making the assumption that they do and responding accordingly.

In addition, this topic is highly compatible with reinforcement learning and observational techniques. IRE could use ground-truth observations at the end of each game to evaluate the links in the selected strategy and adding, strengthening, or even removing them based on what actually occurred. This would allow each strategy to reflect the minor nuances in implementation based on the current meta and hypothetically improve its performance over time.

Similarly, IRE could build entirely new strategies autonomously by creating a strategy tree from the actual assets created in the game and generate a strategy "fingerprint" based on an algorithmic evaluation of said strategy. This could then be added to the strategy library for future use. However, this would cause the strategy library to grow uncontrollably unless an additional algorithm was provided to match the new strategy against existing strategies to prevent duplicates.

Finally, advances in unit micro-management through the use of machine learning would naturally pair with IRE's higher level strategic decision making. More effective individual units would improve IRE's win rate without having any negative impact on its predictive accuracy.

References

Alberto Uriarte, S. O. (2014). *Game-Tree Search over High-Level Game States in RTS Games*. Retrieved from AAIDE: http://nova.wolfwork.com

Champandard, A. (2014, August 12). *Monte-Carlo Tree Search in TOTAL WAR: ROME II's Campaign AI.* Retrieved from Ai Game Dev: http://aigamedev.com/open/coverage/mcts-rome-ii/

He He, J. B.-G. (2016). Opponent Modeling in Deep Reinforcement Learning. *arXiv:1609.05559*.

J. T. Isaacs, e. a. (2014). "GPS-optimal micro air vehicle navigation in degraded environments". *American Control Conference* (pp. pp. 1864-1871). Portland, OR: IEEE.

K. Wendlandt, M. B. (2005). "Indoor localization with probability density functions based on Bluetooth". *IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications, Berlin*, pp. 2040-2044 Vol. 3.

Laviers, K. e. (2009). *Exploiting Early Intent Recognition for Competitive Advantage.* Springfield, VA: KNEXUS RESEARCH CORP.

Markov Decision Process. (2018, 06 12). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Markov_decision_process

Robert Grover Brown, P. Y. (2012). EXPECTATION, AVERAGES, AND CHARACTERISTIC FUNCTION. In *Introduction to Random Signals and Applied Kalman Filtering* (pp. 13-18). John Wiley & Sons, Inc.

S. Venkateswaran, e. a. (2013). "RF source-seeking by a micro aerial vehicle using rotation-based angle of arrival estimates". *American Control Conference* (pp. pp. 2581-2587). Washington, DC: IEEE.

Santiago Ontañon, G. S. (2013, 5 (4)). A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in games, IEEE Computational Intelligence Society*, pp.1-19.

Uriarte, A. (2017). *Adversarial Search and Spatial Reasoning in Real Time Strategy Games.* Philadelphia: Drexel University.