# 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 the NOVA StarCraft competition AI. 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

# 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. 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 is one such bot first developed in 2013 at Drexel University. The Intent Recognition Engine (IRE) builds on top of existing NOVA capabilities to improve its predictive accuracy against enemy bots.

RTS games present intractable 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 solution space that is far, far too large to search using traditional methods. In addition, the real-time nature of the genre means algorithms must act quickly, effectively ruling out many decision-making processes that promise optimality. Developing techniques that can rely only on observable data to make inferences about higher level enemy decision making can help ameliorate this problem by trading optimality for probabilistic predictions. 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 mapped them into a 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.

# 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.



Figure - 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.

* 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 both Chess and even Go in terms of not only state space but also decision tree size.

### StarCraft

StarCraft: Brood War 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.



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

## 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. In StarCraft every unit, building, and research upgrade has a fixed set of pre-requisites that must exist before the object can be produced. When all these components are combined for a given race, it produces a single “tech tree” representing the races’ complete set of capabilities (Figure 3) more fundamentally constrained by the rules below. 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 can be represented as a subtree of the races’ tech tree consisting of only the nodes and edges required to achieve that series of productions. Every strategy in StarCraft can be represented by a build order, and while there is nuance to the exact execution of that build order, such as building location placement or map exploration, the order itself cannot change due to the strict dependencies between objects.

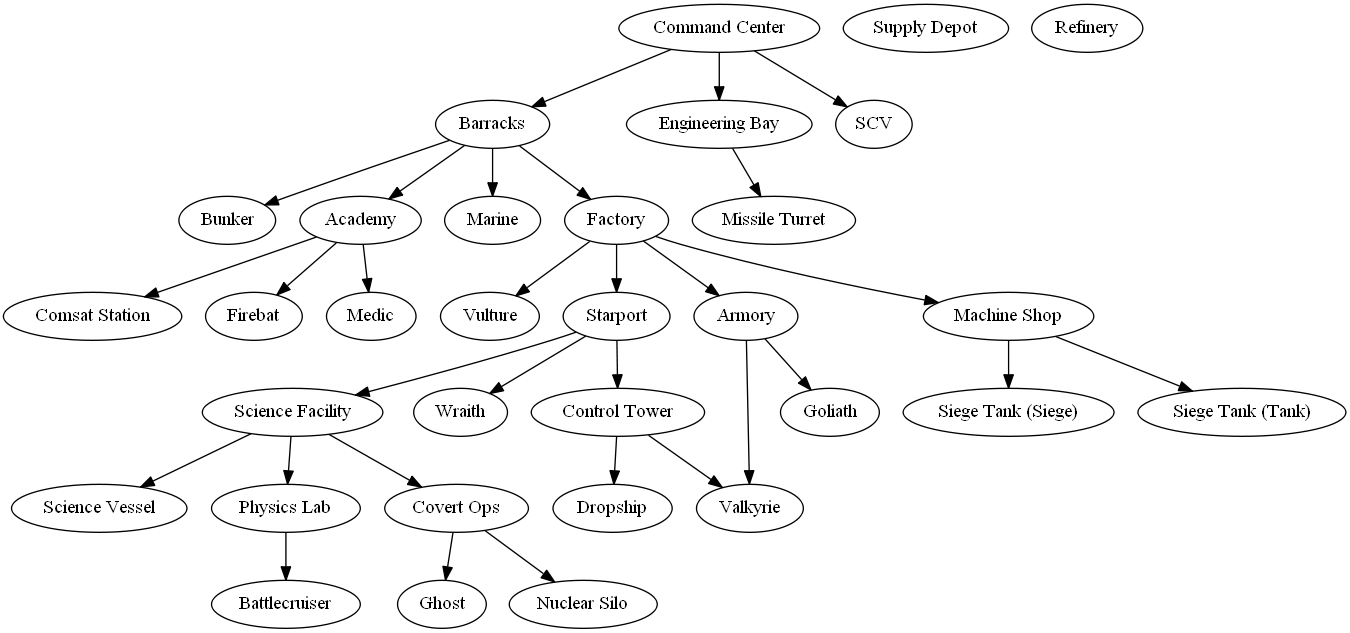


Figure - A segment of the Terran Tech Tree showing some of the dependencies between units and buildings, which are always one-directional.

Strategies can vary in 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. As there are a finite number of unique strategies, there must necessarily be some way to uniquely identify them. By selecting a single set of quantifiable parameters, every strategy can be given a unique “fingerprint” consisting of a measurement for that strategy against each parameter. This complete set of numerical values when combined can be seen as coordinate, not unlike a position in a graph. The coordinate space of this graph is represented as a value along each axis, where each axis is one of the aforementioned quantifiable parameters. Together, this N-dimensional space contains all possible unique combinations of parameters, henceforth called the “strategy space”. In this space, all strategy parameters are normalized from -1 to 1 for consistency. A point in strategy space, then, represents a possible strategy whose parameter intensities match the location provided by the point.

**Formal Representation for the StarCraft Strategies and Strategy Space**

Races in StarCraft: T, P, Z ⊆ R list of all races

n = asset (unit or building)

T, P, Z = ∀n | ∃n ⊆ NT,P,Z = all capabilities for a given race

n0 = root building for each race

Tech Tree T = ∀n ⊆ N | ∃ni ⊆ T | ∑(∃ni ⊆ T) == 1 | ∄nk = n0

Strategy s = s0 = n0 | ∃ni ⊆ T | ∑(∃ni ⊆ T) == 1 | S < T

Strategy Space a ⊆ A axis of al dimensions

S = n dimensional space where ∀n | ∑(∃a ⊆ A) == n

{∀s ⊆ S | ∀n ⊆ s | ∃nix,y,z != ∃njx,y,z}

∃si ⊆ St.p.z | ∄sk ⊆ St.p.z where si == sk

IRE uses a 3-axis system to represent the strategy space. 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 the enemy 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 late-game aerial armada would measure as a -1 on the ground axis, a 1 on the air axis, and a -1 on the aggression axis.

These axes were selected 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 or defending against opponent “rushes” in the hopes of producing more powerful units. These axes were evaluated against the subset of strategies 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, the 18 most common build orders were codified in this manner and scored in the tables below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Terran Strategy Name** | **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 Name** | **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 Name** | **A vs AA** | **G vs AG** | **Atk vs Def** |  |
| 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 |

When a strategy is loaded into strategy space, each node in the subtree is given a 3-dimensional point representing its intensity along those axes. Strategies are normalized to their unique depth, where the lowermost nodes have intensities matching the strategy’s overall values and all other nodes scale evenly between those values and zero. So, a node halfway down a strategy with an aggressiveness of 1 would have an aggressiveness value of 0.5. The pseudocode for adding a strategy to the strategy space is outlined below:

onAddStrategy(NEW strategy)

FORALL nodes in strategy.buildTree

FOREACH axis in strategyAxes

Node.axis = node\_depth / max\_depth \* axis.value

graph\_merge(strategySpace, strategy)

## Probability Density Functions

In order to predict enemy strategies in partial observability, a probabilistic approach must be taken since perfect data will almost always be unavailable. This also benefits early prediction as multiple similar strategies may look nearly identical during early phases of gameplay when the same types of assets are being produced but diverge later when advanced assets 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), or density of a continuous random variable, is a function, whose value at any given sample (or point) in the sample space (the set of possible values taken by the random variable) can be interpreted as providing a relative likelihood that the value of the random variable would equal that sample.[citation needed] In other words, while the absolute likelihood for a continuous random variable to take on any particular value is 0 (since there are an infinite set of possible values to begin with), the value of the PDF at two different samples can be used to infer, in any particular draw of the random variable, how much more likely it is that the random variable would equal one sample compared to the other sample. (Wikipedia)*

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 over time in order 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). 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 an N-dimensional strategy space (Figure 4). Enemy unit or building observations act as “emissions” 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 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.

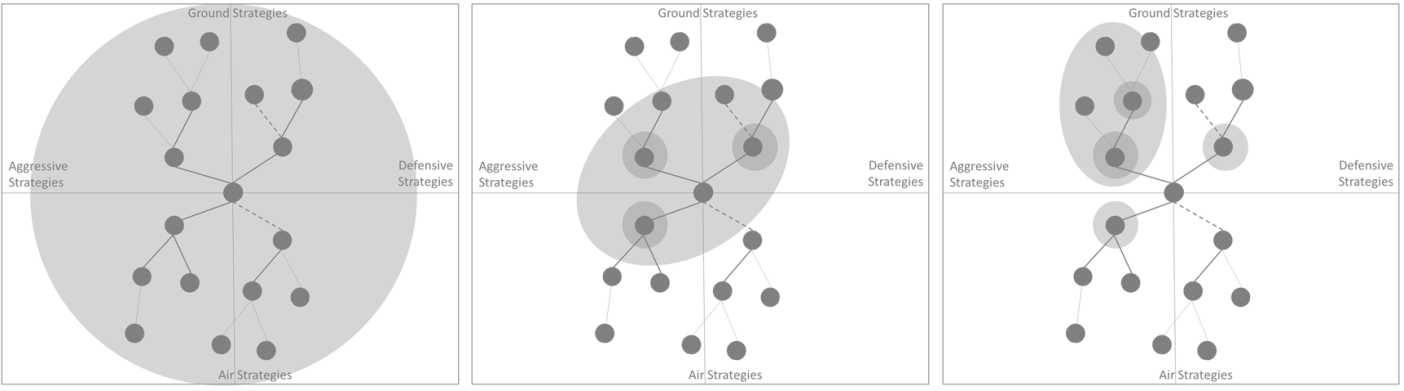


Figure - Initially all strategies are possible. As observations are made, the error bounds around the actual strategy being used converges, eventually settling on the actual strategy being used

# Converting Observations into Predictions

IRE uses the concepts outlined above in conjunction with observed enemy units in order to make predictions about enemy strategies and then, using those predictions, bias NOVA production of units to ensure a proper balance of effective counter-units to maximize effectiveness in battle. IRE does so by integrating into NOVA which is itself integrated with the BWAPI framework. BWAPI interfaces directly with StarCraft to provide data to custom AI, ensuring that during gameplay only data a real player would have access to is provided. IRE relies on existing NOVA command of units and resource management to focus solely on strategy identification and selection (Figure 5). During gameplay, observed enemy units result in a callback into NOVA which handles the micromanagement of responding to the threat. IRE piggybacks off this signal to do its own analysis:

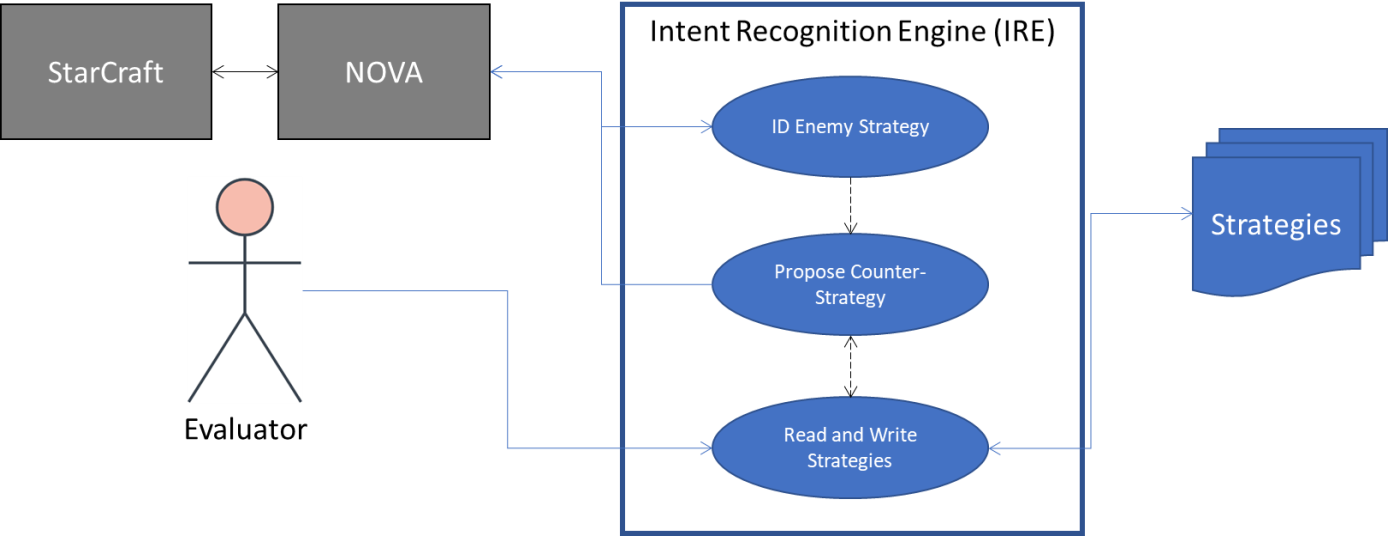


Figure 5 - IRE uses observed enemy units to infer the opponent's possible strategies and use the most likely candidates to bias unit production towards effective counter-units

onObserveEnemyAsset(NEW observedAsset)

FORALL Strategies

FORALL nodes

If(node == observedAsset)

node.intensity++

Each observation may increase the intensity of multiple nodes across different strategies, but as observations are laid on top of each other, the number of potential overlapping strategies decreases. With enough observations, eventually one strategy will emerge as the most likely. This gradually increasing predictive accuracy means that IRE and NOVA can begin producing a mix of relevant counter-units which converges on the most effective units as then number of possible strategies decreases. If the enemy changes strategies early in production, those new observations will converge on a different part of strategy space as the nodes in that strategy overcome the intensities of old nodes. Producing units takes time and resources, and as such 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.

After each observation, IRE makes a prediction as to which strategy the enemy is attempting. It does so by taking the highest probability strategies and averaging their values across each axis. Early in the game, IRE may find itself averaging air and ground strategies and determine that the best response is a balanced counter-strategy of units capable of fighting both. As the game progresses, though, and strategies begin to coalesce at one end of the axis, this will cause IRE to recommend higher production of specific counter-units relative to other unit types. A pseudocode example of how IRE performs these actions is as follows:

onDecideStrategy()

SORT(strategies.nodes, node.intensity, greaterThan)

FOREACH axis in strategyAxes

FOR top 5 nodes in sortedNodes

totalAxisValue += Node.axis.value

axisAverage = totalAxisValue / 5

suggestedCounter = axisAverage \* -1

setBuildPriority(axis, suggestedCounter)

# 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, many hundreds of trials were performed across different enemy types. 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.

(NEED enough details to allow others to replicate)

## 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 without decreasing existing capabilities. 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 concept but could also generalize to other StarCraft or RTS AI. To measure this, we rely on the Jaccard Index of the system (Figure 6), 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.

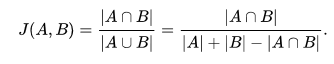


Figure - The Jaccard Index measures the overall predictive accuracy of IRE against enemy strategy selection

## Evaluation Procedure/Setup

All evaluations were run using BWAPI and a multi-instance scripting process that allows for repeated faster than real-time games to be played without human intervention. For evaluation, all trial runs log at completion not only the winner, but also the final game state for each player which is used as ground truth to compare against predictions.

Using this test harness, NOVA was pitted against the standard StarCraft AI set to select a random race in 1000 matches: 500 with just NOVA and 500 with NOVA enhanced by IRE. 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 X% across all trials. NOVA with IRE providing additional support by comparison had an average win ratio of Y%.

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

<Detail performance on a per-race basis using graphics>

The overall predictive accuracy of IRE was XYZ%.

<pie charts of enemy strategies selected across all runs, one for each race>

# 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.

(request permission from anyone specifically for graphics)

## Opponent modeling

Opponent Modeling is likely the most sophisticated approach among comparable approaches in terms of implementation. It uses game theoretics to reason not only on friendly and enemy actions, but the interations 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.

## 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 as conclusions are based 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.* (Wikipedia) Partially Observable MPDs (POMDPs) extend this concept by assuming that the transitions between states can’t be directly observed and can only be probabilistically guessed. POMDPs have been used for predicting opponent state for many years [CITE] but suffer from extremely slow performance due to the massive number of possible states in complex scenarios over time.

To overcome this computation hurdle, research more recently in 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).

# Conclusions

TBD once results are generated

# 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:

* Notes about what didn’t quite work out

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

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.

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.