Predicting Army Combat Outcomes in StarCraft

Marius Stanescu, Sergio Poo Hernandez,
Graham Erickson, Russel Greiner and **Michael Buro**
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2E8
{astanesc|pooherna|gkericks|rgreiner|mburo}@ualberta.ca

Abstract

Smart decision making at the tactical level is important for Artificial Intelligence (AI) agents to perform well in the domain of real-time strategy (RTS) games. This paper presents a Bayesian model that can be used to predict the outcomes of isolated battles, as well as predict what units are needed to defeat a given army. Model parameters are learned from simulated battles, in order to minimize the dependency on player skill. We apply our model to the game of StarCraft, with the end-goal of using the predictor as a module for making high-level combat decisions, and show that the model is capable of making accurate predictions.

1 Introduction

1.1 Purpose

Real-Time Strategy (RTS) games are a genre of video games in which players must gather resources, build structures from which different kind of troops can be trained or upgraded, recruit armies and command them in battle against opponent armies. RTS games are an interesting domain for Artificial Intelligence (AI) research because they represent well-defined complex adversarial systems and can be divided into many interesting sub-problems (Buro 2004). The current best RTS game-playing AI still performs quite poorly against human players. Therefore, the research community is focusing on developing RTS agents to compete against other RTS agents (Buro and Churchill 2012). For the purpose of experimentation, the RTS game StarCraft[1] is currently the most common platform used by the research community, as the game is considered well balanced, has a large online community of players, and has an open-source interface (BWAPI[2]).

Usually, in a RTS game, there are several different components a player needs to master in order to achieve victory. One such sub-problem is the combat scenario (usually called a battle). Each player has a known quantity of each type of unit (called an *army*) and is trying to defeat the opponent's army while keeping his own units alive. Combat is an important part of playing RTS games, as winning battles will affect the outcome of a match. We consider *1*) given two specific armies (each composed of a specified set of units of each type), predicting which will win; and *2*) given one army, specify what other army is most likely to defeat it.

[1] http://en.wikipedia.org/wiki/StarCraft

[2] http://code.google.com/p/bwapi/

1.2 Motivation

One successful framework for developing AI for the combat aspect of RTS games relies on alpha-beta search, where nodes are evaluated by estimating the combat outcome of two specified armies (Buro and Churchill 2012) – i.e., the winning player, assuming that the two players fight until one of them has no units left. One standard way to predict the outcome of such a combat is to use a simulator, where the behavior of the units is determined by deterministic scripts (e.g., attack closest unit) (Churchill, Saffidine, and Buro 2012). This is time intensive, especially as the number of units grows. The model we propose is inspired by rating systems such as the ELO ratings used in Chess (Elo 1978) or TrueSkill™ (Herbrich, Minka, and Graepel 2007). By learning from battle outcomes between various combinations of units, one can predict the combat outcome of such a battle using a simple mathematical equation, rather than time consuming simulations. We are particularly interested in large scale battles, which would take a longer time to solve using such simulators.

A second challenging problem is determining an army that will have a good chance of defeating some other specified army. Given some unit combination controlled by the opponent, we want to know similar sized armies that can probably defeat it. A system that can answer this type of questions could be used by tactical planners to decide which units should be built and what unit combinations should be sent to battle the opponent. For example, if an important location (e.g. near a bridge or choke-point) is controlled by the opponent with X units (X_1 of type 1, X_2 of type 2, etc), what unit combination Y should be sent to defeat X? Even more, what properties would such a combination Y require? Maybe it needs firepower above a certain threshold, or great mobility. Currently, there are no systems that can answer such questions quickly and accurately.

1.3 Objectives

The objectives of this work are to develop a model that can answer these two types of questions effectively. Accuracy for the battle prediction problem (i.e. given two armies, who wins?) can be measured by the effectiveness of a trained model on match-ups set aside for testing. Accuracy for the most likely army problem (i.e. given one army, what other army is most likely to defeat it?) can be measured using play-out scripts (i.e. when fixed policies are used, does the army predicted to win, actually win?). From now on we will refer to the first question as *who wins*, and the second question as *what wins*.

The next section presents background on Bayesian Networks as applied to RTS games, and on ranking systems. In section 3 we introduce our proposed model, and explain the process of learning the model parameters. In section 4, we present the data we use to evaluate the model, along with several experiments and a discussion of the results. Future extensions and conclusions are discussed in the section 5.

2 Background

2.1 Bayesian Networks

The model we present uses Bayesian techniques in the form of a Bayesian network, which is a type of Probabilistic Graphical Model (PGM) (Koller and Friedman 2009). Bayesian Networks represent a set of random variables (which could be observable quantities, latent variables or unknown parameters) and their conditional dependencies using a directed acyclic graph, whose edges denote conditional dependencies. Each node has an associated probability function that, for each specific assignment to the node's direct parents, gives a probability distribution of the variable represented by that node.

Bayesian networks encode the uncertainty of the situation, allowing incompleteness (in a formal logic sense) to be transformed into uncertainty about the situation (Jaynes 2003). There is much uncertainty in RTS games: the opponent's state is only partially known, moves are made simultaneously, and RTS games represent complex, dynamic environments that are difficult to model completely. Even more, PGMs allow us to develop a single model that can support different types of queries. Using variables that have known values as "*evidence*", we can inquire about the (conditional) probability distribution of the remaining variable(s).

Naturally, PGMs have seen an increased popularity in the RTS domain over the past few years. Hidden Markov Models (a simple type of PGM) have been used to learn high-level strategies from data (Dereszynski et al. 2011) and PGMs have been used to predict the opponent's opening strategy (Synnaeve and Bessiere 2011b) and to guess the order that the opponent is building units in Synnaeve and Bessiere (2011a). The same research group has also developed models that allow their RTS agent to make decisions about where on the map it should send units to attack and with what kinds of units (Synnaeve and Bessiere 2012a). The model makes major simplifications about unit types and does not allow the agent to ask questions about how many of a specific unit type it should produce. Synnaeve and Bessiere (2011c) have also used Bayesian modeling to enable units to be individually reactive. This allows each unit to behave individually when navigating the map; moreover, during combat the unit will determine if it needs to retreat or continue fighting. The model only works for individual unit decisions, but is not used to predict an outcome between two armies, which is one of our interests.

Most recently, the same research group has clustered armies based on their unit compositions and shown how battle predictions can be made using the cluster labels (Synnaeve and Bessiere 2012b), which effectively tackles the *who wins* problem. However, their method was shown to have a low prediction accuracy, and differs from ours in two ways. First, their model was developed on replay data (instead of simulated data) which adds the noise of different player skills to the problem. Second, their model can only be used to predict the following question: given two armies, which one wins? It cannot be used to predict an army that can defeat the opponent.

We chose to use a Bayesian network for our model, as there are many advantages to such a choice. First, we could learn the model once, then answer both types of questions simply by performing different types of queries on the same model. Secondly, defining the model structure (based on intuition and domain knowledge) helps to simplify the learning and the inference tasks. Finally, Bayesian networks allow uncertainty to be modeled explicitly – i.e. predictions are reported in form of likelihoods or probabilities.

2.2 Rating Systems

A problem similar to battle outcome prediction is rating/ranking – the task of attaching some numeric quantities to subjects, then arranging these subjects in a specific order, consistent with the assigned values. Rating systems have been historically used to estimate a player's skill in one-on-one matches (Elo 1978), and subsequent systems even measured the uncertainty of that estimation (Glickman 1999).

During the last decade, rating systems have been extended for events that include more players. The most significant results were obtained by TopCoder's ranking algorithm[3] and by Microsoft's approach, TrueSkill™ (Herbrich, Minka, and Graepel 2007). The probabilistic model used by TrueSkill™ is designed to deal with players that take part in games or enter tournaments, and compete in teams of various sizes. It estimates the skills of these players after each match (or competition). The system is initialized with a prior one dimensional Gaussian distribution over each player's skills: $s \sim N(\mu, \sigma^2)$. The mean player's *true skill* is μ, where σ indicates the uncertainty of the prior. After k games, the posterior can be computed, and we obtain μ^k, σ^k, where this σ^k will usually decrease with the number of matches, as we get more information about the respective player. This approach could prove very useful, if we treat a battle as a match between an arbitrary number of units on each side. Then, after observing a number of battles, we would have skill estimates for each unit type. As each army is a "team"

[3] http://apps.topcoder.com/wiki/display/tc/Algorithm+Competition+Rating+System

of units, we can combine the skills of all units in a team/army – for example, by adding them up – and use this sum to predict the outcome of any future battle.

3 Proposed Model and Learning Task

Current rating systems associate a single latent variable to each person (his skill). This may work well for predicting the outcome of a chess match, but units in RTS games are inherently different and the outcome of a battle depends on features such as damage, attack range, hit points, armor or speed. Consequently, we need a model with multiple latent features for every unit. Besides being able to predict battle outcomes, such a model could also provide insight into why an army defeats another (e.g. army A wins because it has very high damage output while army B is lacking in the hit points attribute).

3.1 The Model

Using a unit's hit point and attack values, Churchill, Saffidine, and Buro (2012) propose the following evaluation function for combat games, based on the life-time damage a unit can inflict:

$$\text{LTD} = \sum_{u \in U_A} \text{Hp}(u)\text{Dmg}(u) - \sum_{u \in U_B} \text{Hp}(u)\text{Dmg}(u)$$

U_A and U_B are the units controlled by player A and B; $\text{Hp}(u)$ and $\text{Dmg}(u)$ are the hit points and damage the unit u inflicts per second. This was shown to be effective and could serve as a starting point for our model. Furtak and Buro (2010) prove that in 1 vs. n units combat scenarios, there is an optimal way for the lone unit to choose its targets: to minimize its sustained damage, it should order its targets by decreasing value of $\text{Dmg}(u)/\text{Hp}(u)$.

We would like to use a similar formula to predict whether army A can win against army B. Of course, since an army has more units, we need to define composite features such as Hp(A), Hp(B), Dmg(A), Dmg(B), where for example $\text{Hp(A)} = \sum_{u \in U_A} \text{Hp}(u)$. We can then define the quantity:

$$\text{Dmg(A)}/\text{Hp(B)} - \text{Dmg(B)}/\text{Hp(A)},$$

This expression will directly influence the probability of army A winning against army B. The winning probability for army A will be higher if the offensive feature – Dmg(A) – is high, and the opponent's defense – Hp(B) – is low. Our intuition is that combining terms from both armies (such as damage of first army over hit points of second army) should work well. Moreover, it allows us to generalize from damage and hit point to offensive and defensive features. That way, we could integrate into our model interactions such as damage type vs. armor, or army mobility vs. attack range by using a general term such as $(O_A/D_B - O_B/D_A)$. We assume all features to have strictly positive values. A similar idea worked very well in Stanescu (2011).

The resulting graphical model is shown in Figure 1. We will briefly explain each type of node. At the top we have a discrete node for each type of unit, representing the number of units (0-100) of that type in an army (node C = unit **C**ount). There are *U* different unit types, and so U instances

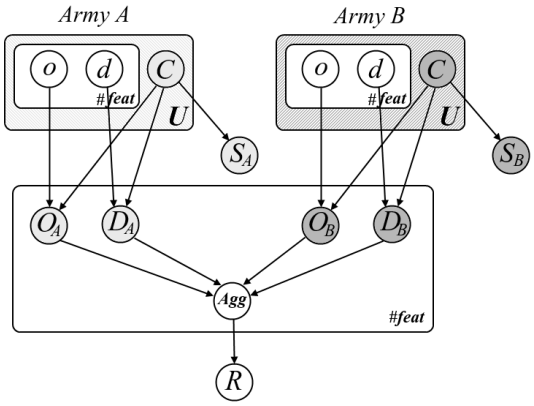


Figure 1: Proposed graphical model.

of this plate for each army (for *plate notation* see Buntine (1994, p. 16)); there are two plates (left, right) that represent the two different armies in the modeled network. Next, for every unit there are *#feat* tuples of features, each having an offensive (node o) and defensive (node d) attribute, as described above (for a total of *2*#feat* features). For example, some offensive/defensive tuples could be {damage, hit points} or {attack range, mobility}. These nodes are one dimensional Gaussian variables.

Then, for every feature, we compute the aggregate offensive (node O) and defensive (node D) features for each army by adding all individual unit features: $O = \sum_{i=1}^{U} C_i \cdot o_i$. This is of course a simple model; here we explore whether it is sufficient. Next, for each feature, we combine the aggregate offense and defense into a single node, using formula previously introduced:

$$\text{Agg}_i = (O_{A_i}/D_{B_i} - O_{B_i}/D_{A_i}).$$

There will be a total of *#feat* such nodes, which we anticipate to provide the information needed to determine the outcome of the battle. This last node (R - **R**esult) provides a value $\in [0, 1]$, corresponding to the probability that army A will defeat army B. For this, we use a sigmoid function (Han and Moraga 1995) of the sum of all combined nodes:

$$R = \frac{1}{1 + e^{-\sum_{i=1}^{F} \text{Agg}_i}}$$

There is one remaining node type (S - **S**upply), which enforces supply restrictions on armies. Supply is a value StarCraft uses to restrict the amount of units a player can command: each unit has a supply value (e.g. marine 1, zealot 2, dragoon 2, etc.), and an army's supply value is the sum of its composing units' supply values. We incorporate supply into our model to avoid trivial match-ups (e.g. 20 marines will defeat 1 marine), and to be able to ask for armies of a specific size.

We decided to start with this simple but potentially effective model structure, and focus on learning the features in an efficient manner. In future work we could investigate more complex models, and other ways of combining the features (rather than a simple sum).

3.2 Learning

Having specified the structure of our model, we need to learn the offensive and defensive feature values for each unit type (nodes at the top of our graphical model). Afterwards, we can start asking queries and predicting battle outcomes.

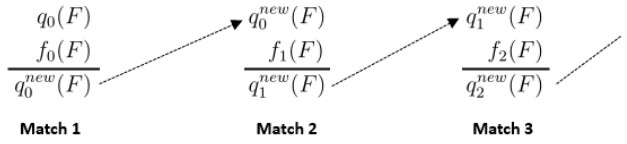
Let $M = (m_1, \dots m_N)$ be the results of the observed matches, where each m_k is either A or B, depending on who won. We let $F = (\text{feat}_i^j)$ denote the vector of features for all unit types; feat_i^j is feature i $\in \{1, 2, \dots 2 * \#feat\}$ of unit type j (out of U unit types). We will learn one such vector $F' = \arg\max P(M, F)$ based on observing the data set M, as we assume that the unit features stay constant and do not change between matches (eg. a marine will always have the same damage, speed or hit points). The joint distribution of F and the N results of the matches is then

$$P(M, F) = P(F) \prod_i P(m_i|F) \ (m_j \text{ is the result of match } j).$$

Because exact inference is intractable (at one point we will need to compute integrals of sigmoid functions times Gaussians), we will use the core approximation technique employed by (Herbrich, Minka, and Graepel 2007) in TrueSkill™ - Gaussian density filtering (GDF). This method, also known as *moment matching* or *online Bayesian learning*, is commonly used for approximating posteriors in Bayesian models (Minka 2001). Given a joint distribution over some observed variables M and hidden parameters F, it computes a Gaussian approximation q of the posterior $P(F|M)$:

$$q(F) \sim N(\mu, \sigma).$$

To use Gaussian density filtering, we need to factor the joint distribution into a product of factors $P(M, F) = \prod_i f_i$. We can choose $f_0 = p(F)$ as the prior and $f_i(F) = p(m_i|F)$ as the other factors, one for each battle. We use the prior to initialize the posterior, and step through all the factors, updating and incorporating each one into our posterior. At every step we start with a Gaussian belief about the feature vector F, which is our current approximation $q(F)$. We update it based on the new observation's likelihood $f_i(F)$ to obtain an approximate posterior $q_i^{new}(F)$:



The exact posterior, which is difficult to compute, is

$$\hat{P}_i(F) = \frac{f_i(F)q(F)}{\int_F f_i(F)q(F)\, dF}.$$

We find the approximate posterior $q_i^{new}(F)$ by minimizing the KL divergence: $q_i^{new}(F) = \arg\min_q \text{KL}(\hat{P}_i(F)||q)$, while requiring that it must be a Gaussian distribution (Minka 2001). This reduces to moment matching, hence the alternative name for this method. The $q_N^{new}(F)$ obtained after processing all factors is the final approximation we will use in our model.

4 Experiments

Because we are most interested in comparing armies in terms of the units that compose them, we made several simplifications of an RTS battle. Terrain or advantages caused by terrain are not considered by the model. Spell-casters and flying units are also left out. Upgraded units and upgrades at the per-unit level are not taken into account. Battles are considered to be independent events that are allowed to continue until one side is left with no remaining units. That is, we do not represent reinforcements or retreating in our model, and the outcome of one battle is unrelated to the outcome of another battle. Furthermore, only one-on-one battles (in terms of one player versus another player) are modeled explicitly.

4.1 Data

For the prediction problem (*who wins?*), the model's input is in the form of tuples of unit counts. Each player is represented by one tuple, which has an element for each unit type. For the current version of the model, only four different unit types are considered (here two Terran faction units - marines and firebats, and two Protoss faction units - zealots and dragoons). For each unit type, the value of the element is the number of units of that type in the player's army. The tuples refer to the armies as they were at the start of the battle. The output of the model is a soft prediction (a probability of one of the players winning).

The input to the model for the most likely army problem (*what wins?*) is similar. Two army tuples are given, but the values in some or all of the elements of one of the armies are missing (including the supply value, which, if specified, can be used as a restriction). The output is an assignment for the missing values that corresponds to the army most likely to win the battle.

For training and testing purposes, we generated data-sets using a StarCraft battle simulator (*SparCraft*, developed by David Churchill, UAlberta[4]). The simulator allows battles to be set up and carried out according to deterministic play-out scripts (or by decision making agents, like an adversarial search algorithm). We chose to use simulated data (as opposed to data taken from real games) because the simulator allows us to produce a large amount of data (with all kinds of different unit combinations) and to avoid the noise caused by having players of different skill and style commanding the units. This would be an interesting problem to investigate, but it is outside of the scope of this paper.

The simulations use deterministic play-out scripts. Units that are out of attack range move towards the closest unit, and units that can attack target the opponent unit with the highest damage-per-second to hit-point ratio. This policy was chosen based on its success as an evaluation policy in search algorithms (Churchill, Saffidine, and Buro 2012). Two data-sets were created: *1*) a data-set of armies of ten supply (33 different armies, 1089 different battles), and *2*) a data-set of armies of fifty supply (153 different armies, 23409 different battles).

The simulator does not have unit collision detection, which means that we can position all units of an army at

[4] https://code.google.com/p/sparcraft/

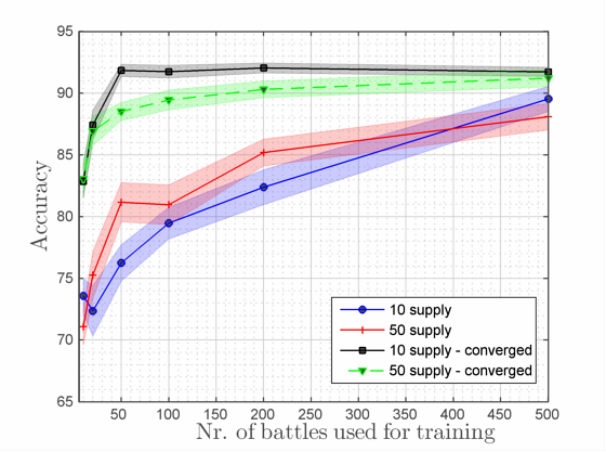


Figure 2: Accuracy results for *GDF – one pass* and *GDF – until convergence*, for **who wins** experiments.

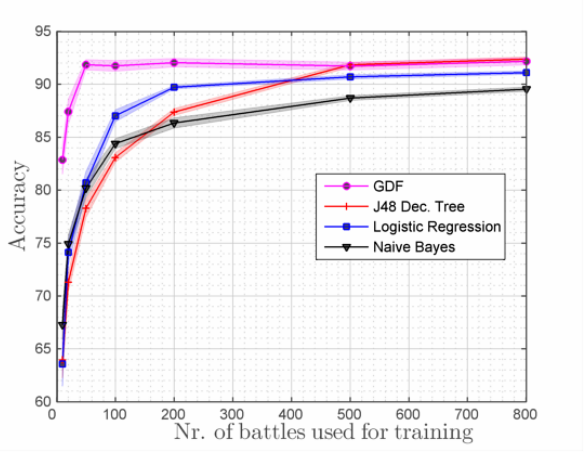


Figure 3: Accuracy results comparing GDF and three standard classifiers, for *who wins* experiments, 10 supply battles.

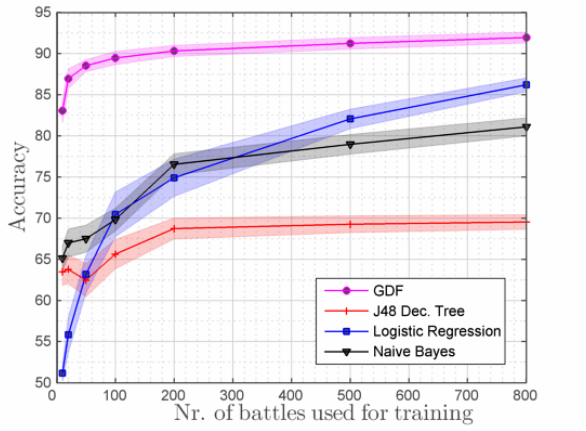


Figure 4: Accuracy results comparing GDF and three standard classifiers, for *who wins* experiments, 50 supply battles.

the same position on the map. Using this option, we can generate two data sets for each supply limit. One data set has the armies at opposite sides of the map in a line formation. This was an arbitrary choice and it is not affecting the model's parameters. The other data set has all units of each army in a single fixed position at opposite sides of the map. We explored how using the single fixed position affects the accuracy of the model.

We are interested in answering two main questions: **who wins?** - given two armies, which one is more likely to win? and **what wins?** - given an opponent army, what army do we need to build in order to defeat it? Both questions will be tested on the 10 supply and 50 supply data sets.

4.2 Who Wins

This section describes the experiments we ran to answer the *who wins* question. First, we are interested in the model's capability to generalize, and how it performs given a number of battles for training. If there is good performance even for a low number of battles, then we can train against a specific opponent, for example when playing StarCraft AI tournaments. We randomly chose 10, 20, 50, 100, 200, 500 and 800 battles for training, and 500 other battles to predict as a test. The accuracy is determined by how many outcomes the model is able to predict correctly. We compute the mean accuracy for 20 experiments, and show error bars (shaded area) for one standard error on either side of the mean.

Minka (2001) notes that more passes of GDF on the same (training) data leads to significantly improved results, a tendency also confirmed by Stanescu (2011). Consequently, after we process the data once, we run the same algorithm again using our approximation as a starting point. We repeat this until convergence – when the difference in successive approximations falls under a certain threshold. In our case, around 10 iterations were enough to reach the best performance. We show the results in Figure 2.

The more training data we use the better the model performs, which is what we expected. The results are very encouraging; the improvement brought by several GDF passes is obvious, and training on more than 50 battles provides almost no additional gain. In the following experiments we use this version of the algorithm, as the results are significantly better than only one GDF pass.

We compare our method with several popular algorithms, as a baseline. We use logistic regression, a naive Bayes classifier, and J48 decision trees, all of which are implemented in Weka using default parameters (Hall et al. 2009). We chose these classifiers because they are well-known and simple to implement. The results are shown in Figure 3 (for 10 supply armies) and in Figure 4 (for 50 supply armies).

For 10 supply armies, all algorithms have a similar performance for training with large data sets (800 battles). However, the lower the number of battles, the better our algorithm does, in comparison. After training on 20 battles, the accuracy is better than any of the other algorithms, trained on 100 battles. When increasing the size of the armies to 50 supply, the results are even better. Even training with only 20 battles, we achieve well over 85% accuracy, better than any of the baseline algorithms do even after seeing 800 battles!

Finally, in Table 1 we compare different starting positions of the armies: line formation vs. all units juxtaposed at the same fixed position. In addition, a 10 supply data set was

Table 1: Accuracy of *who wins* experiment, using different starting positions for 10 (upper section) and 50 supply armies (lower section).

		Number of battles in training set					
Pos.	S	10	20	50	100	200	500
Line		82.83%	87.38%	91.81%	91.71%	92.00%	91.69%
Fixed		82.39%	85.96%	86.75%	89.10%	89.94%	90.00%
Fixed	✓	82.71%	87.37%	91.57%	90.84%	91.02%	90.77%
Line		83.03%	86.90%	88.48%	89.41%	90.27%	91.18%
Fixed		81.94%	86.29%	85.99%	85.12%	86.02%	85.02%

created that uses a search algorithm as the play-out policy instead of simple scripted policies. This provides better accuracy than its scripted counterpart, probably because the fights are played by *stronger* players. For the juxtaposed armies, the accuracy drops drops by 1-2% compared to the spread formations. We conjecture this may be because clustering range units together provides them with an increasing advantage, as the size of the armies grow. They would be able to focus fire and kill most of the oncoming opponents instantly, and they would be far more valuable in high numbers. Note, however, that our current system is not able to model this, as it computes the army features as a sum of all individual features. The model is not even aware of the unit positions, as the only inputs are the number and type of units, along with the winner.

4.3 What Wins

For this question we first created all possible Protoss and Terran army combinations (of two unit types) that can be built with 10 and 50 supply (44 and 204 combinations, respectively). Once trained (using the same data as the *who wins* experiment), we provide each (known) army combination as inputs. Then, the model will predict the Protoss army that is most likely to defeat (with 90% probability) the known, given army. Sometimes, if the given army is very strong, the model is not able to provide an army to defeat it. Each given answer is then tested with the simulator to verify that the predicted army wins, as expected. The accuracy of the model is measured by the percentage of instances where the predicted army actually wins.

The model has two options for predicting armies:

- With an army supply limit, the predicted army supply size must be equal to the limit.
- Without an army supply limit, the predicted army can be of any size.

The results are shown in the Table 2. We see that the model is able to predict a correct army that wins against an army of 50 supply 97% of the time, when no supply limit is specified. With the supply limit enabled, the accuracy drops to 87%. Against armies of 10 supply it is less accurate in both cases, and drops again, from 82% to 68%. Predicting winning 10 supply armies is the hardest task, because there are a few very strong armies, making it hard to defeat with the limited available choices of 10 supply (5 Protoss units). Therefore, the model is unable to predict an army that wins with 90% probability in almost 22% of the cases.

Table 2: Accuracy of *what wins* experiment, with and without supply limits for 10 and 50 supply armies.

		Predicted army		
Data	Supply limit	Wins	Loses	Unavailable
10 supply		82.5%	17.5%	0%
10 supply	✓	68.2%	9.1%	22.7%
50 supply		97.9%	0%	2.1%
50 supply	✓	87.3%	8.8%	3.9%

It is clear that with no supply limit the system is able to accurately predict an army that can win, which might happen because there are more available armies to choose from, most of which have larger supply than the opponent (and are clearly advantaged). The closer we ask the model to match the known army in supply, the worse it is at estimating a winning army, because the battles are more evenly matched.

5 Conclusions and Future Work

The results we obtain are very promising. Our model can very accurately predict *who wins*. The model does not do as well in estimating *what wins* but the results are positive and it might be just a matter of modifying the model by adding features that work better at estimating the answer for this question. Moreover, trying more complex ways of combining the features (rather than a simple sum) should lead us to a better understanding of the problem, and could further increase the accuracy of the model.

A StarCraft agent would greatly benefit from incorporating such a framework. Accurately predicting *who wins* could be used to avoid fighting battles against superior armies, or to determine when flow of a battle is against the player and the units should be retreated. However, sometimes losing units could potentially be strategically viable, if the armies are close in strength but the opponent's army was more expensive (either resource cost or training time). Estimating *what wins* would be used to decide what units to build in order to best counter the opponent's army. Furthermore, it could also help the build order module by making suggestions about the units needed in the future.

These are problems often encountered in the real-time strategy domain, and consequently our model would prove useful in most RTS games. We also think that it could potentially transfer to other games such as multiplayer online battle arena (MOBA) games (eg. DOTA[5]). By recording the starting hero for each of the ten players (who fight in a 5 vs. 5 match) and the outcome of the game, we could potentially find out which heroes are stronger than others (overpowered) for balancing reasons. A system that makes recommendations for which heroes to chose at the start of the game is a

[5]https://en.wikipedia.org/wiki/Defense_of_the_Ancients

viable option, too.

Furthermore, we are exploring several extensions:

- Currently the model is using 6 features for each unit type. We plan to increase the number of features to explore how this affects the accuracy for both questions. It would be interesting to see if adding more features increases accuracy or if it leads to overfitting.
- We are also adding more army constraints. Currently we are using the supply cost of the units in the army, but another way of limiting the army size is by the resource cost of training that army. Resources are one of the most popular metrics used to determine the army strength by other systems that work with StarCraft. Being able to enforce both supply and resource constraints would prove very useful.
- Our current model deals with only 4 unit types (marines, firebat, zealots and dragoons). We would like to expand it to use other unit types available in StarCraft. This change is critical if we want the model to be used as part of a StarCraft playing AI.
- We would like to represent additional information, such as the units' positions or number of hit points. Currently we treat all units as "new" and having maximum life, which is not always the case.
- We work with a simulated data set, which makes everything easier by disregarding several aspects of the game such as unit collision and the noise induced by human micromanagement skills. We would like to compare our model on *real data* (for example extracted from game replays), which is a more difficult task.
- One limitation for the current model is assuming that units have independent contributions to the battle outcome. This may hold for a few troop types, but is particularly wrong when considering units such as spell-casters, which promote interactions with other units using spells. We also miss taking into account combinations such as melee+ranged units, which are more efficient in general. We need to either consider a few smart features to take these into account, or add correlations between different types of units.
- Finally, we want to expand the questions the model can answer. We want to investigate how good the model is at estimating – given some already-built units – what units we need to add to our army in order to defeat our opponent. Expanding the model should be straightforward, as we could simply duplicate the number of unit count nodes: for each type of unit, have a node for the number of already existing units, and one for units that could potentially be built. This would also work for representing the opponent, making the difference between observed or unobserved units.

References

Buntine, W. L. 1994. Operations for learning with graphical models. *arXiv preprint cs/9412102*.

Buro, M., and Churchill, D. 2012. Real-Time Strategy game competitions. *AI Magazine* 33(3):106.

Buro, M. 2004. Call for AI research in RTS games. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 139–142.

Churchill, D.; Saffidine, A.; and Buro, M. 2012. Fast heuristic search for RTS game combat scenarios. *Proceedings of AIIDE,(pre-print available at www. cs. ualberta. ca/mburo/ps/aiide12-combat. pdf*.

Dereszynski, E.; Hostetler, J.; Fern, A.; Dietterich, T.; Hoang, T.-T.; and Udarbe, M. 2011. Learning probabilistic behavior models in real-time strategy games. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Elo, A. E. 1978. *The rating of chessplayers, past and present*, volume 3. Batsford London.

Furtak, T., and Buro, M. 2010. On the complexity of twoplayer attrition games played on graphs. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*.

Glickman, M. E. 1999. Parameter estimation in large dynamic paired comparison experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 48(3):377–394.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter* 11(1):10–18.

Han, J., and Moraga, C. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *From Natural to Artificial Neural Computation*. Springer. 195–201.

Herbrich, R.; Minka, T.; and Graepel, T. 2007. Trueskill ™: A Bayesian skill rating system. *Advances in Neural Information Processing Systems* 19:569.

Jaynes, E. T. 2003. *Probability theory: the logic of science*. Cambridge university press.

Koller, D., and Friedman, N. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Minka, T. P. 2001. *A family of algorithms for approximate Bayesian inference*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Stanescu, M. 2011. Rating systems with multiple factors. Master's thesis, School of Informatics, Univ. of Edinburgh, Edinburgh, UK.

Synnaeve, G., and Bessiere, Pierre, e. a. 2011a. A Bayesian model for plan recognition in RTS games applied to StarCraft. *Proceedings of AIIDE* 79–84.

Synnaeve, G., and Bessiere, P. 2011b. A Bayesian model for opening prediction in RTS games with application to StarCraft. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 281–288. IEEE.

Synnaeve, G., and Bessiere, P. 2011c. A Bayesian model for RTS units control applied to StarCraft. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 190–196. IEEE.

Synnaeve, G., and Bessiere, P. 2012a. Special tactics: A Bayesian approach to tactical decision-making. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 409–416. IEEE.

Synnaeve, G., and Bessiere, P. e. a. 2012b. A dataset for StarCraft AI & an example of armies clustering. In *Artificial Intelligence in Adversarial Real-Time Games: Papers from the 2012 AIIDE Workshop AAAI Technical Report WS-12-15*.