Using Lanchester Attrition Laws for Combat Prediction in StarCraft

Marius Stanescu, Nicolas Barriga, and Michael Buro
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2E8
{astanesc|barriga|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. Winning battles is crucial in RTS games, and while ahumans can decide when and how to attack based on their experience, it is challenging for AI agents to estimate combat outcomes accurately.

A few existing models address this problem in the game of StarCraft but present many restrictions, such as not modeling injured units, supporting only a small number of unit types, or being able to predict the winner of a fight but not the remaining army. Prediction using simulations is a popular method, but generally slow and requires extensive coding to model the game engine accurately.

This paper introduces a model based on Lanchester's attrition laws which addresses the mentioned limitations while being faster than running simulations. Unit strength values are learned using maximum likelihood estimation from past recorded battles. We present experiments that use a StarCraft simulator for generating battles for both training and testing, and show that the model is capable of making accurate predictions. Furthermore, we implemented our method in a StarCraft bot that uses either this or traditional simulations to decide when to attack or to retreat. We present tournament results (against top bots from 2014 AIIDE competition) comparing the performances of the two versions, and show increased winning percentages for our method.

Introduction

A Real-Time Strategy (RTS) game is a video game in which players gather resources and build structures from which different types of units can be trained or upgraded in order to recruit armies and command them into battle against opposing armies. RTS games are an interesting domain for Artificial Intelligence (AI) research because they represent well-defined complex adversarial environments and can be divided into many interesting sub-problems (Buro 2004). Current state of the art AI systems for RTS games are still not a match for good human players, but the research community is hopeful that by focusing on RTS agents to compete against other RTS agents we will soon reach the goal of defeating professional players (Ontanón et al. 2013)

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 features an open-source programming interface (BWAPI [2]).

RTS games contain different elements aspiring players need to master. Possibly the most important such component is *combat* in which each player controls an army (consisting of different types of units) and is trying to defeat the opponent's army while minimizing its own losses. Winning such battles has a big impact on the outcome of the match, and as such, combat is a crucial part of playing RTS games proficiently. However, while human players can decide when and how to attack based on their experience, it is challenging for current AI systems to estimate combat outcomes.

(Churchill and Buro 2012) estimate the combat outcome of two armies for node evaluation in their alpha-beta search which selects combat orders for their own troops. Similarly, (Stanescu, Barriga, and Buro 2014b; 2014a) require estimates of combat outcomes for state evaluation in their hierarchical search framework and use a simulator for this purpose. Even if deterministic scripted policies (e.g., "attack closest unit") are used for generating unit actions within the simulator (Churchill, Saffidine, and Buro 2012), this process is still time intensive, especially as the number of units grows.

(Stanescu et al. 2013) recognize the need for a fast prediction method for combat outcomes and propose a probabilistic graphical model that, after being trained on simulated battles, can accurately predict winners. While being a promising approach, there are several limitations that still need be addressed:

- the model is linear in the unit features, i.e. the offensive score for a group of 10 marines is ten times the score for 1 marine. While this could be accurate for close-ranged (melee) fights, it severely underestimates being able to focus fire in ranged fights (this will be discussed in depth later)
- their model deals with only 4 unit types so far, and scaling it up to all StarCraft units might induce training problems such as overfitting and/or accuracy reduction

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

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

- it only predicts the winner but not the remaining army size
- all units are treated as having maximum hit points, which does not happen often in practice; there is no support for partial hit points
- all experiments are done on simulated data, and
- correlations between different unit types are not modeled.

In this paper we introduce a model that addresses the first five limitations listed above, and we propose an extension (future work) to tackle the last one. Such extensions are needed for the model to be useful in practice, e.g. for speeding up hierarchical search and adjusting to different opponents by learning unit strength values from past battles.

We proceed by first discussing current combat game state evaluation techniques and Lanchester's battle attrition laws. We then show how they can be extended to RTS games and how the new models perform experimentally in actual StarCraft game play. We finish with ideas on future work in this area.

Background

As mentioned in the previous section, the need for a fast prediction method for combat outcomes has already been recognized (Stanescu et al. 2013). The authors propose a probabilistic graphical model that, after being trained on simulated battles, can accurately predict the winner in new battles. Using graphical models also enables their framework to output unit combinations that will have a good chance of defeating some other specified army (i.e., given one army, what other army of a specific size is most likely to defeat it?).

We will only address the first problem here: single battle prediction. We plan to use our model for state evaluation in a hierarchical search framework similar to those described in (Stanescu, Barriga, and Buro 2014a) and (Uriarte and Ontañón 2014). Consequently, we focus on speed and accuracy of predicting the remaining army instead of only the winner. Generating army combinations for particular purposes is not a priority, as the search framework will itself produce different army combinations which we only need to evaluate against the opposition.

Similarly to (Stanescu et al. 2013), our model will learn unit feature weights from past battles, and will be able to adjust to different opponents accordingly. We choose maximum likelihood estimation over a Bayesian framework for speed and simplicity. Incorporating Lanchester equations in a graphical model would be challenging, and any complex equation change would require the model to be redesigned. The disadvantages are that potentially more battles will be needed to reach similar prediction accuracies and batch training must be used instead of incrementally updating the model after every battle.

There are several existing limitations our model will address:

- better representation of ranged weapons by unit group values depending exponentially on the number of units instead of linearly,
- including all StarCraft unit types,
- adding support for partial hit points for all units involved in the battle, and
- predicting the remaining army of the winner.

Lanchester Models

The seminal contribution of Lanchester to operations research is contained in his book "Aircraft in Warfare: The Dawn of the Fourth Arm" (Lanchester 1916). He starts by justifying the need for such models with an example: consider two identical forces of 1000 men each; the Red force is divided into two units of 500 men each which serially engage the single (1000 man) Blue force. According to the Quadratic Lanchester model (introduced below), the Blue force completely destroys the Red force with only moderate loss (e.g., 30%) to itself, supporting the "concentration of power" axiom of war that states that forces are not to be divided. The possibility of equal or nearly equal armies fighting and resulting in relatively large winning forces are one of the interesting aspects of war simulation based games.

Lanchester equations represent simplified combat models: each side has identical soldiers, and each side has a fixed strength (no reinforcements) which governs the proportion of enemy soldiers killed. Range, terrain, movement, and all other factors that might influence the fight are either abstracted within the parameters or ignored entirely. Fights continues until the complete destruction of one force (which Lanchester calls a "conclusion"). The equations are valid until one of the army sizes is reduced to 0.

Lanchester's Linear Law is given by the following differential equations:

$$\frac{dA}{dt} = -\beta AB \quad \text{and} \quad \frac{dB}{dt} = -\alpha BA \ ,$$

where t denotes time and A, B are the force strengths (number of units) of the two armies assumed to be functions of time. Parameters α and β are attrition rate coefficients representing how fast a soldier in one army can kill a soldier in the other. The equation is easier to understand if one thinks of β as the relative strength of soldiers in army B; it influences how fast army A is reduced. The pair of differential equations above may be combined into one equation by removing time as a variable:

$$\alpha(A - A_0) = \beta(B - B_0) \ ,$$

where A_0 and B_0 represent the initial forces. This is called a *state solution* to Lanchester's differential equation system that does not explicitly depend on time). The origin of the term *linear law* is now apparent because the last equation describes a straight line.

Lanchester's Linear Law applies when one soldier can only fight one other soldier at a time. If one side has more soldiers some of them won't always be fighting as they wait for an opportunity to attack. In this setting, the casualties suffered by both sides are proportional to the number actually fighting and the attrition rates. If $\alpha = \beta$, then the above example of splitting a force into two and fighting the enemy sequentially will have the same outcome as without splitting:

a draw. This was originally called *Lanchester's Law of Ancient Warfare*, because it is a good model for battles fought with edge weapons.

Lanchester's Square Law is given by:

$$\frac{dA}{dt} = -\beta B \quad \text{and} \quad \frac{dB}{dt} = -\alpha A \; .$$

In this case, the state solution is

$$\alpha(A^2 - {A_0}^2) = \beta(B^2 - {B_0}^2) \; .$$

Increases in force strength are more important than for the linear law, as we can see from the concentration of power example. The squared law is also known as *Lanchester's Law of Modern Warfare* and is intended to apply to ranged combat, as it quantifies the value of the relative advantage of having a larger army. However, the squared law has nothing to do with range – what is really important is the rate of acquiring new targets. Having ranged weapons generally lets your soldiers engage targets as fast as they can shoot, but with a sword or a pike to which the Linear Law applies one would have to first locate a target and then move to engage it.

The general form of the attrition differential equations is:

$$\frac{dA}{dt} = -\beta A^{2-n} B \quad \text{and} \quad \frac{dB}{dt} = -\alpha B^{2-n} A \; ,$$

where n is called the *attrition order*. We have seen previously that for $n = 1$, the resulting attrition differential equations give rise to what we know as Lanchester's Linear Law, and to the Lanchester's Square Law for $n = 2$. As expected, the state solution is

$$\alpha(A^n - {A_0}^n) = \beta(B^n - {B_0}^n) \; .$$

The exponent which is called attrition order represents the advantage of a higher rate of target acquisition and applies to the size of the forces involved in combat, but not to the fighting effectiveness of the forces which is modeled by attrition coefficients α and β. The higher the attrition order, the faster any advantage an army might have in combat effectiveness is overcome by numeric superiority. This is the equation we use in our model, as our experiments suggest that for StarCraft battles an attrition order of ≈ 1.56 works best on average, if we had to choose a fixed order for all possible encounters.

The Lanchester Laws we just discussed have several limitations we need to overcome to apply them to RTS combat, and some extensions (presented in more detail in the following section) are required:

- we must account for the fact that armies are comprised of different RTS game unit types and
- currently soldiers are considered either dead or alive, while we need to take into account that RTS game units can enter the battle with any fraction of their maximum hit points.

Lanchester Model Extensions for RTS Games

The state solution for the Lanchester general law can be rewritten as

$$\alpha A^n - \beta B^n = \alpha {A_0}^n - \beta {B_0}^n = k \; .$$

The constant k depends only on the initial army sizes A_0 and B_0. Hence, for prediction purposes, if $\alpha {A_0}^n > \beta {B_0}^n$ then P_A wins the battle. If we note A_f and B_f to be the final army sizes, then $B_f = 0$ and $\alpha {A_0}^n - \beta {B_0}^n = \alpha {A_f}^n - 0$ and we can predict the remaining victorious army size A_f.

To use the Lanchester laws in RTS games, a few extensions have to be implemented. Firstly, it is rarely the case that both armies are composed of a single unit type. We therefore need to be able to model heterogeneous army compositions. To this extent, we replace army effectiveness α with an average value α_{avg}. Assuming that army A is composed of N types of units, then

$$\alpha_{avg} = \frac{\sum_{i=1}^{N} a_i \alpha_i}{A} = \frac{\sum_{j=1}^{A} \alpha_j}{A} \; ,$$

where A is the total number of units, a_i is the number of units of type i and α_i is their combat effectiveness. Alternatively, we can sum over all individual units directly, α_j corresponding to unit j.

Consequently, predicting battle outcomes will require a combat effectiveness (we can also call it unit strength for simplicity) for each unit type involved. We start with a default value

$$\alpha_i = \text{dmg}(i)\text{HP}(i) \; ,$$

where dmg(i) is the unit's damage per frame value and HP(i) its maximum number of hit points. Later, we aim to learn $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots\}$ by training on recorded battles.

The other necessary extension is including support for injured units. Let us consider the following example: army A consists of one marine with full health, while army B consists of two marines with half the hit points remaining. Both the model introduced by (Stanescu et al. 2013) and the life-time damage (LTD) evaluation function proposed by (Kovarsky and Buro 2005)

$$\text{LTD2} = \sum_{u \in U_A} \text{HP}(u)\text{dmg}(u) - \sum_{u \in U_B} \text{HP}(u)\text{dmg}(u)$$

would mistakenly predict the result as a draw. The authors also designed the life-time damage-2 (LTD2) function which departs from linearity by replacing HP(u) with $\sqrt{\text{HP}(u)}$ and will work better in this case.

In the time a marine deals damage equal to half its health, army B will kill one of army A's marines, but would also lose his own unit, leaving army A with one of the two initial marines intact, still at half health. The advantage of focusing fire becomes even more apparent if we generalize to n marines starting with $1/n$ health versus one healthy marine. Army A will only lose one of its n marines, assuming all marines can shoot at army B's single marine at the start of the combat. This lopsided result is in stark contrast to the predicted draw.

Let's model this case using Lanchester type equations. Denoting the attrition order with o, the combat effectiveness of a full health marine with m and that of a marine with $1/n$ health as m_n, we have:

$$n^o m_n - 1^o m = (n-1)^o m_n \implies m_n = \frac{m}{n^o - (n-1)^o}$$

If we choose an attrition order between the linear ($o = 1$) and the square ($o = 2$) laws, $o = 1.65$ for example, then $m_2 = m/2.1383$, $m_3 = m/2.9887$ and $m_4 = m/3.7221$. Intuitively picking the strength of an injured marine to be proportional with its current health $m_n = m/n$ is close to these values, and would lead to extending the previous strength formula for an individual unit like so:

$$\alpha_i = \text{dmg}(i)\text{HP}(i) \cdot \frac{\text{currentHP}(i)}{\text{HP}(i)} = \text{dmg}(i)\text{currentHP}(i).$$

Learning Combat Effectiveness

For predicting the outcome of combat C between armies A and B we first compute the estimated army remainder score μ_C using the generalized Lanchester equation:

$$\mu_C = \alpha_C A^o - \beta_C B^o$$

From army A's perspectives μ is a positive value if army A wins, 0 in case of a draw, and negative otherwise. As previously mentioned, experiments using simulated data suggest that $o = 1.56$ yields the best accuracy, if we had to choose a fixed order for all possible encounters. Fighting the same combat multiple times might lead to different results depending on how players control their units, and we choose a Gaussian distribution to model the uncertainty of the army remainder score r:

$$P_C(r) = N(r; \mu_C, \sigma^2) ,$$

where σ is a constant chosen by running experiments. Just deciding which player survives in a small scale fight where units can't even move is PSPACE-hard in general (Furtak and Buro 2010). Hence, real-time solutions require approximations and/or abstractions. Choosing a Gaussian distribution for modeling army remainder score is a reasonable candidate which will keep computations light.

Let us now assume that we possess data in the form of remaining armies A_f and B_f (either or both can be zero) from a number of combats $\boldsymbol{C} = \{C_1, C_2, \ldots, C_n\}$. A datapoint C_i consists of starting army values A_i, B_i and final values A_{if}, B_{if}. We compute the remainder army score R_i using the Lanchester equation:

$$R_i = \alpha_{C_i} A_{if}^o - \beta_{C_i} B_{if}^o$$

This enables us to use combat results for training even if no side is dead by the end of the fight.

Our goal is to estimate the effectiveness values α_i and β_i for all encountered unit types and players. The distinction needs to be made, even if abilities of a marine are the same for both players. If the player in charge of army A is more proficient at controlling marines then α_{marine} should be higher than β_{marine}.

The likelihood of $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ given $\boldsymbol{C}$ and $\boldsymbol{R} = \{R_1, R_2, \ldots, R_n\}$ is used for approximating the combat effectiveness; the maximum likelihood value can then be chosen as an estimate. The computation time is usually quite low using conjugate gradients, for example, and can potentially be done once after several games or even at the end of a game.

If we assume that the outcomes of all battles are independent of each other and the probability of the data given the combat effectiveness values is

$$P(\boldsymbol{R}|\boldsymbol{C}, \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}) = \prod_i N(R_i; \mu_{C_i}, \sigma^2) ,$$

then we can express the log likelihood

$$\mathcal{L}(\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}) = \sum_i \log N(R_i; \mu_{C_i}, \sigma^2) .$$

The maximum likelihood value can be approximated by starting with some default parameters, and optimizing iteratively until we are satisfied with the results. We use a gradient ascent method, and update with the derivatives of the log likelihood with respect to the combat effectiveness values. Using a Gaussian distributions helps us to keep the computations manageable.

To avoid overfitting we modify the error function we are minimizing by using a regularization term:

$$Err = -\mathcal{L}(\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}) + \gamma Reg(\{\boldsymbol{\alpha}, \boldsymbol{\beta}\})$$

If we want to avoid large effectiveness values for example, we can pick $Reg = \sum_i \alpha_i^2 + \sum_i \beta_i^2$. We chose

$$Reg = \sum_i (\alpha_i - d_i)^2 + \sum_i (\beta_i - d_i)^2 ,$$

where d_i are the default values computed in the previous subsection using basic unit statistics. In the experiments section we show that these estimates already provide good results. The γ parameter controls how close the trained effectiveness values will be to these default values.

Experiments and Results

To test the effectiveness of our models in an actual RTS game (StarCraft) we had to simplify actual RTS game battles. Lanchester models do not take into account terrain features that can influence battle outcomes. In addition, upgrades of individual units or unit types are not yet considered, but could later be included using *new*, virtual units (e.g., a dragoon with range upgrade is a different unit than a regular dragoon). However, that would not work for regular weapon/armor upgrades, as the number of units would increase beyond control. For example, the Protoss faction has 3 levels of weapon upgrades, 3 of shields and 3 of armor, so considering all combinations would add 27 versions for the same unit type.

Battles are considered to be independent events that are allowed to continue no longer than 800 frames (around 30 seconds game time), or until one side is left with no remaining units, or until reinforcements join either side. Usually StarCraft battles do not take longer than that, except if there is a constant stream of reinforcements or one of the players keeps retreating, which is difficult to model.

Experiments Using Simulator Generated Data

We start by testing the model with simulator generated data, similarly to (Stanescu et al. 2013). The authors use four different unit types (marines and firebats from the Terran faction, zealots and dragoons from the Protoss faction), and individual army sizes of up to population size 50 (e.g., marines count 1 towards the population count, and zealots and dragoons count 2, etc.). For our experiments, we add three more unit types (vultures, tanks and goliaths) and we increase the population size from 50 to 100.

The model input consists of two such armies, where all units start with full health. The output is the predicted remaining army score of the winner, as the loser is assumed to fight to death. We compare against the *true* remaining army score, obtained after running the simulation.

For training and testing at this stage, we generated battles using a StarCraft battle simulator (*SparCraft*, developed by David Churchill, UAlberta[3]). The simulator allows battles to be set up and carried out according to deterministic play-out scripts, or by search-based agents. We chose the simple yet effective *attack closest* script, which moves units that are out of attack range towards the closest unit, and attacks units with the highest damage-per-second to hit point ratio. This policy, which was also used in (Stanescu et al. 2013), was chosen based on its success as an evaluation policy in search algorithms (Churchill, Saffidine, and Buro 2012). Using deterministic play-out scripts eliminates noise caused by having players of different skill or playing style commanding units.

We randomly chose 10, 20, 50, 100, 200, and 500 different battles for training, and a test set of 500 other battles to predict outcomes. The accuracy is determined by how many outcomes the model is able to predict correctly. We show the results in Table 1, where we also include corresponding results of the Bayesian model from (Stanescu et al. 2013) for comparison. The datasets are not exactly the same, and by increasing the population limit to 100 and the number of unit types we increase the problem difficulty.

The results are very encouraging: our model outperforms the Bayesian predictor on a more challenging dataset. As expected, the more training data, the better the model performs. Switching from 10 battles to 500 battles for training only increases the accuracy by 3.3%, while in the Bayesian model it accounts for a 8.2% increase. Moreover, for training size 10 the Lanchester model is 6.8% more accurate, but just 2% better for training size 500. Our model performs better than the Bayesian model on small training sizes because we start with already good approximations for the unit battle strengths, and the regularization prevents large deviations from these values.

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

Table 1: Accuracy of Lanchester and Bayesian models, for different training sets sizes. Testing was done by predicting outcomes of 500 battles in all cases. Values are winning percent averages over 20 experiments.

	Number of battles in training set					
Model	10	20	50	100	200	500
Lanchester	89.8	91.0	91.7	92.2	93.0	93.2
Bayesian	83.0	86.9	88.5	89.4	90.3	91.2

Experiments in Tournament Environment

Testing on simulated data validated our model, but ultimately we need to assess its performance in the actual environment it was designed for. For this purpose, we integrated the model into UAlbertaBot, one of the top bots in recent AIIDE StarCraft AI competitions [4]. UAlbertaBot is an open source project for which detailed documentation is available online[5]. The bot uses simulations to decide if it should attack the opponent with the currently available units — if a win is predicted — or retreat otherwise. We replaced the simulation call in this decision procedure by our model's prediction.

UAlbertaBot's strategy is very simple: it only builds zealots, a basic Protoss melee unit, and tries to rush the opponent and then keeps the pressure up. This is why we do not expect large improvements from using Lanchester models, as they only help to decide to attack or to retreat. More often than not this translates into waiting for an extra zealot or attacking with one zealot less. This might make all the difference in some games, but using our model to decide what units to build, for example, could lead to bigger improvements. In future work we plan to integrate this method into a search framework and a build order planner such as (Köstler and Gmeiner 2013).

When there is no information on the opponent, the model uses the default unit strength values for prediction. Six top bots from the last AIIDE competition[6] take part in our experiments: IceBot (1^{st}), LetaBot (3^{rd}), Aiur (4^{th}), Xelnaga (6^{th}), original UAlbertaBot (7^{th}), and MooseBot (9^{th}). Ximp (2^{nd}) was left out because we do not win any games against it in either case. It defends its base with photon cannons (static defense), then follows up with powerful air units which we cannot defeat with only zealots. Skynet (5^{th}) was also left out because against it UAlbertaBot uses a hard-coded strategy which bypasses the attack/retreat decision and results in a 90% win rate.

Three tournaments were run: 1) our bot using simulations for combat prediction, 2) using the Lanchester model with default strength values, and 3) using a new set of values for each opponent obtained by training on 500 battles for that particular match-up. In each tournament, our bot plays 200 matches against every other bot.

To train model parameters, battles were extracted from the previous matches played using the default weights. A battle is considered to start when any of our units attacks or is attacked by an enemy unit. Both friendly and opposing units close to the attacked unit are logged with their current health as the starting state of the battle. Their health (0 if dead) is recorded again at the end of the battle — when any of the following events occurs:

- one side is left with no remaining units,
- new units reinforce either side, or
- 800 frames have passed since the start of the battle.

There are some instances in which only a few shots are fired and then one of the players keeps retreating. We do not

[4]http://webdocs.cs.ualberta.ca/∼cdavid/starcraftaicomp/
[5]https://github.com/davechurchill/ualbertabot/wiki
[6]http://webdocs.cs.ualberta.ca/∼scbw/2014/

Table 2: Tournament results against 6 top bots from AIIDE 2014 competition. Three tournaments are played, with different options for the attack/retreat decision. In the first our bot uses simulations, and in the second the Lanchester model with default strength values. In the third we use battles from the second tournament for training and estimating new strength values. Winning percentages are computed from 200 games for each match-up, 20 each on 10 different maps.

	UAB	Xelnaga	Aiur	MooseBot	IceBot	LetaBot	**Avg.**
Sim.	60.0	79.0	84.0	65.5	19.5	57.0	**60.8**
Def.	64.5	81.0	80.5	69.0	22.0	66.5	**63.9**
Train	69.5	78.0	86.0	93.0	23.5	68.0	**69.7**

consider such cases as proper battles. For training we require battles in which both players keep fighting for a sustained period of time. Thus, we removed all fights in which the total damage was less than 80 hit points (a zealot has 160 for reference) and both sides lost less than 20% of their starting total army hit points. We obtained anywhere from 5 to 30 battles per game, and only need 500 for training.

Results are shown in Table 2. Our UAlbertaBot version wins 60% of the matches against the original UAlbertaBot because we have updated the new version with various fixes that mainly reduce the number of timeouts and crashes, especially in the late game.

On average, the Lanchester model with learned weights wins 6% more games than the same model with default strength values, which is still 3% better than using simulations. It is interesting to note that the least (or no) improvement occurs in our best match-ups, where we already win close to 80% of the games. Most of such games are lopsided, and one or two extra zealots do not make any difference to the outcome. However, there are bigger improvements for the more difficult match-ups, which is an encouraging result.

The only exception is IceBot, which is our worst enemy among the six bots we tested against. IceBot uses bunkers to defend which by themselves do not attack but can load up to four infantry units which receive protection and a range bonus. We do not know how many and what infantry units are inside, and the only way to estimate this is by comparing how much damage our own units take when attacking it. These estimates are not always accurate, and furthermore, IceBot also sends workers to repair the bunkers. Consequently, it is very difficult to estimate strength values for bunkers, because it depends on what and how many units are inside, and if there are workers close by which can (and will) repair them. Because UAlbertaBot only builds zealots and constantly attacks, if IceBot keeps the bunkers alive and meanwhile builds other, more advanced units, winning becomes impossible. The only way is to destroy the bunkers early enough in the game. We chose to adapt our model by having five different combat values, one for empty bunker (close to zero), and four others for bunker with one, two, three or four marines inside. We still depend on good damage estimations for the loaded units and we do not take into account that bunkers become stronger when repaired, which is a problem we would like to address in future work.

Conclusions and Future Work

In this paper we have introduced and tested generalizations of the original Lanchester models to adapt them to making predictions about the outcome of RTS game combat situations. We also showed how model parameters can be learned from past combat encounters, allowing us to effectively model opponents' combat strengths and weaknesses. Pitted against some of the best entries from a recent StarCraft AI competition, UAlbertaBot with its simulation based attack-retreat code replaced by our Lanchester equation based prediction, showed encouraging performance gains.

Because even abstract simulations in RTS games can be very time consuming, we speculate that finding accurate and fast predictors for outcomes of sub-games – such as choosing build orders, combat, and establishing expansions – will play an important role in creating human-strength RTS game bots when combined with look-ahead search. Following this idea, we are currently exploring several model extensions which we briefly discuss in the following paragraphs.

A common technique used by good human players is to snipe off crucial or expensive enemy units and then retreat, to generate profit. This is related to the problem of choosing which unit type to target first from a diverse enemy army, a challenge not addressed much in current research. Extending the current Lanchester model from compounding the strength of all units into an average strength to using a matrix which contains strength values for each own unit versus each opponent unit might be a good starting point. This extension would enable bots to kill one type of unit and then retreat, or to deal with a unit type which is a danger to some of its other units. For instance, in some cases ranged units are the only counter unit type to air units and should try to destroy all fliers before attacking ground units.

A limitation of the current model is assuming that units have independent contributions to the battle outcome. This may hold for a few unit types, but is particularly wrong when considering units that promote interactions with other units, such as medics which can heal other infantry, or workers that can repair bunkers. We could take some of these correlations into account by considering groups of units as a new, virtual unit and trying to estimate its strength.

Another limitation of our prediction model is that it completely ignores unit positions, and only takes into account intrinsic unit properties. An avenue for further research is to expand the model to take into account spatial information, possibly by including it into the combat effectiveness values.

Lastly, by comparing the expected outcome and the real result of a battle, we could possibly identify mistakes either we or the opponent made. AI matches tend to be repetitive, featuring many similar battles. Learning to adjust target priorities or to change the combat scripts to avoid losing an early battle would make a big difference.

References

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., and Buro, M. 2012. Incorporating search algorithms into RTS game agents. In *AIIDE Workshop on Artificial Intelligence in Adversarial Real-Time Games*.

Churchill, D.; Saffidine, A.; and Buro, M. 2012. Fast heuristic search for RTS game combat scenarios. In *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*.

Furtak, T., and Buro, M. 2010. On the complexity of two-player attrition games played on graphs. In Youngblood, G. M., and Bulitko, V., eds., *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010*.

Köstler, H., and Gmeiner, B. 2013. A multi-objective genetic algorithm for build order optimization in StarCraft II. *KI-Künstliche Intelligenz* 27(3):221–233.

Kovarsky, A., and Buro, M. 2005. Heuristic search applied to abstract combat games. *Advances in Artificial Intelligence* 66–78.

Lanchester, F. W. 1916. *Aircraft in warfare: The dawn of the fourth arm*. Constable limited.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game AI research and competition in StarCraft. *TCIAIG* 5(4):293–311.

Stanescu, M.; Barriga, N. A.; and Buro, M. 2014a. Hierarchical adversarial search applied to real-time strategy games. In *Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Stanescu, M.; Barriga, N. A.; and Buro, M. 2014b. Introducing hierarchical adversarial search, a scalable search procedure for real-time strategy games. In *European conference on Artificial Intelligence*.

Stanescu, M.; Hernandez, S. P.; Erickson, G.; Greiner, R.; and Buro, M. 2013. Predicting army combat outcomes in StarCraft. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Uriarte, A., and Ontañón, S. 2014. Game-tree search over high-level game states in RTS games. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.