Learning Probabilistic Behavior Models in Real-time Strategy Games

Ethan Dereszynski and **Jesse Hostetler** and **Alan Fern** and **Tom Dietterich**
Thao-Trang Hoang and **Mark Udarbe**
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, Oregon 97331

Abstract

We study the problem of learning probabilistic models of high-level strategic behavior in the real-time strategy (RTS) game *StarCraft*. The models are automatically learned from sets of game logs and aim to capture the common strategic states and decision points that arise in those games. Unlike most work on behavior/strategy learning and prediction in RTS games, our data-centric approach is not biased by or limited to any set of preconceived strategic concepts. Further, since our behavior model is based on the well-developed and generic paradigm of hidden Markov models, it supports a variety of uses for the design of AI players and human assistants. For example, the learned models can be used to make probabilistic predictions of a player's future actions based on observations, to simulate possible future trajectories of a player, or to identify uncharacteristic or novel strategies in a game database. In addition, the learned qualitative structure of the model can be analyzed by humans in order to categorize common strategic elements. We demonstrate our approach by learning models from 331 expert-level games and provide both a qualitative and quantitative assessment of the learned model's utility.

Introduction

Models of player behavior in real-time strategy (RTS) domains are of significant interest to the AI community. Good models of behavior could improve automated agents, for example by augmenting the strategy representations used in some architectures (Aha, Molineaux, and Ponsen 2005; Ontañón et al. 2007) or guiding the Monte-Carlo simulations of an opponent (Chung, Buro, and Schaeffer 2005; Balla and Fern 2009). They could be incorporated into intelligent assistants that help human players reason about the state of the game and provide predictions about an opponent's future actions. They could also be used in the analysis of game play, to automatically identify common strategic elements or discover novel strategies as they emerge.

In this paper, we focus on learning probabilistic models of high-level strategic behavior and the associated task of strategy discovery in the RTS game *StarCraft*. By "strategy," we mean a player's choice of units and structures to build, which dictates the tone of the game. Our models are learned automatically from collections of game logs and capture the temporal structure of recurring strategic states and decision points. Importantly, our models facilitate the use of general probabilistic reasoning techniques, which makes them directly applicable to any of the tasks mentioned above. In particular, in this paper we demonstrate that our models can be used to categorize strategic play, identify uncharacteristic strategies, and make predictions about a player's future actions and the progression of future game states.

The most obvious use of a strategy model is for *strategy prediction*. The objective is to use features of the game state to predict the opponent's future actions. Several researchers have studied strategy prediction. Schadd, Bakkes, and Spronck (2007) developed a hierarchical opponent model in the RTS game *Spring*. At the top level, players were classified as either "aggressive" or "defensive" based on the frequency of attacks. At the bottom level, players were classified into specific strategies by applying handcoded rules to the observed counts of the opponent's units. Weber and Mateas (2009) examined strategy prediction in *StarCraft* using supervised learning techniques to classify an opening build order into a set of handcrafted categories.

To build a predictive strategy model, one first has to define the possible strategies. The utility of the model depends heavily on the degree to which the chosen strategy labels are informative. In the prediction work described so far, the choice of labels was made by the designers, drawing on their knowledge of the game. A potential weakness of handcrafted labels is that they may be biased toward strategies that are well-known or easy to describe, rather than those that have high predictive or strategic value. They can also be vague, failing to capture the variation in the behavior they are describing. For example, the label "rushing" is often used to describe early aggression in RTS games, but the timing and composition (number and types of military units) of the aggression varies widely between games, and demands different counter-strategies. To be useful for informing gameplay, a strategy model must make predictions about the specific threats that a player is likely to face.

In contrast to the manual specification of labels, *strategy discovery* seeks to learn a set of labels by revealing recurring patterns in gameplay data. This data-driven approach avoids the potential biases of engineered labels. On the contrary, it has the potential to expand the understanding of

strategic play and even recognize novelties. Relatively little work has been done in this direction. Perhaps most similar to our approach, Hsieh and Sun (2008) represent strategies as paths through a lattice. Nodes in the lattice correspond to counts of different units, buildings, and researched technologies. Using hundreds of *StarCraft* games, the authors learn a transition model between nodes (i.e., the next unit or building to be constructed given the current state). Although this model represents technology dependencies and build orders nicely, it cannot predict the *timing* of future events (Weber and Mateas 2009) because it does not model time.

Our approach to strategy discovery models a player's strategy as a sequence of hidden states that evolves over time. Each state encodes a set of preferences for building units and structures of different types. At regular time intervals, the player can move from one state to another according to a set of transition probabilities. The building preferences of each state and the probabilities of transitioning between states are learned from the data. Specific strategies manifest themselves as high-probability trajectories through the states. Our approach is distinguished by the combination of three key attributes. First, we learn our strategy vocabulary directly from data. Second, our model incorporates time, allowing us to predict *when* future events will occur and to use knowledge of the timing of observed events to inform our beliefs. Third, because we use a probabilistic model, we can formulate the prediction task as probabilistic inference, which allows us to quantify the uncertainty in the answers.

The remainder of this paper is organized as follows. In the next section, we introduce our representation of the game state and strategies, describe our encoding of the game state as a Hidden Markov Model, and explain how this model is learned from data. Then we evaluate our model on *Starcraft* gameplay logs and produce a qualitative analysis of the learned model. We interpret the learned states in the context of well-known *Starcraft* strategies and evaluate our model's predictive performance on *StarCraft* games. We conclude with some directions for future work in this domain.

Representation and Modeling

At each point in time, we model the player as being in one of K possible states. Each state has an associated set of preferences for what types of units to construct. As the player plays the game, he or she is modeled as moving from one state to another and building units according to the states that he or she visits. Hence, we can describe the player's strategy as a trajectory through a state space.

We divide the game into a sequence of 30-second intervals, and we summarize each interval t by a binary observation vector $O^t = (O^t_1, \ldots, O^t_U)$, where U is the total number of types of units ("Zealot", "Reaver", "Cybernetics Core", etc.), and O^t_u is 1 if at least one unit of type u was constructed during interval t and 0 otherwise. In this first investigation, we focus on modeling the initial seven minutes of each game, so there are 14 time steps (and observation vectors) per game. We use only the first seven minutes because in the early game, players execute their strategies in relative isolation, whereas later in the game, actions are increasingly

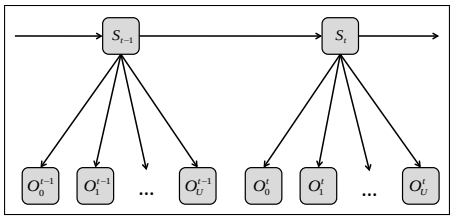

Figure 1: Two-slice representation of the HMM. Squares indicate random variables and arrows denote a conditional relationship between variables.

dictated by tactical considerations such as the composition of the opponent's army or the outcomes of key battles.

An important quality of our model is that the hidden states and transitions between them are not defined in advance. Instead, we apply statistical learning to discover the set of states that best explains the data. Because the model is not sufficient to capture all aspects of *StarCraft* and because players must randomize their play to avoid being too predictable, we formulate this model probabilistically using the well-known Hidden Markov Model (HMM) formalism (Rabiner 1990). Such a probabilistic model can capture the likelihood of different strategy choices and also the probability that the player will produce particular units in each state.

Figure 1 shows a two-timestep slice of the HMM. The nodes labeled S_{t-1} and S_t represent the player states at times $t-1$ and t respectively and take values in $\{1, \ldots, K\}$. The remaining nodes represent observations as defined above. An arrow from a parent to a child node indicates that the value of the parent probabilistically influences the value of the child. The model captures two types of probabilistic dependencies: the transition probabilities and the observation (or build) probabilities. The transition probability distribution $P(S_t|S_{t-1})$ specifies the probability that the player will make a transition from state S_{t-1} to state S_t. For each possible value of S_{t-1} (i.e., one of $\{1, \ldots, K\}$), this probability is a multinomial distribution, $P(S_t|S_{t-1} = k) \sim \text{Multinomial}(\alpha^k_0, \alpha^k_1, \ldots, \alpha^k_K)$, where $\alpha^k_{k'}$ is the probability of transitioning from state k to state k'. This distribution is time invariant, meaning that it is the same for any value of t and thus does not depend on the absolute time.

The observation distribution $P(O^t|S_t)$ is the probability that we will observe the unit production vector $O^t = (O^t_1, \ldots O^t_U)$ at time t given that the player is in state S_t. We model the production of each unit type as a biased coin (Bernoulli random variable) whose probability of being 1 is denoted by θ^k_u: $P(O^t_u|S_t = k) \sim \text{Bernoulli}(\theta^k_u)$. A Bernoulli distribution captures the distinction between producing or not producing a particular unit. This distinction is generally more informative of a player's strategy than knowing the amount of the unit produced beyond one. The model assumes that the production probabilities of different unit types are conditionally independent given the current state, which implies that the joint observation distribution is just the product of the individual unit probabilities: $P(O^t|S_t = k) = \prod_u P(O^t_u|S_t = k)$. Like the transition distribution, the observation distribution is time-invariant.

To complete the HMM, we define the probability of starting in state k at time $t = 0$. This is modeled as a multino-

mial (K-sided die): $P(S_0) \sim \text{Multinomial}(\beta_0, \beta_1, \ldots, \beta_K)$. Putting everything together, our overall behavior model is described by the concatenation of all model parameters $\Phi = (\alpha_1^1, \ldots \alpha_K^K, \beta_0, \ldots, \beta_K, \theta_1^1, \ldots, \theta_U^K)$.

Probabilistic Inference

Given an HMM, it is possible to efficiently answer many types of probabilistic queries about the model variables. The scope of this paper precludes details of the inference algorithms. However, we can say that most queries of interest, including the ones used in this work, have a time complexity that scales linearly in the sequence length and quadratically in the number of states. Typically a query will be in the context of certain observations, which specify concrete values for some of the variables in the HMM, and the task is to infer information about the values of certain other unobserved variables. For example, a predictive query may take the form $P(O_u^{t+d} = 1 | O^0, O^1, \ldots, O^t)$ for $d = 1, \ldots, T - t$, which can be interpreted as, "Given what I have seen up to time t, what is the probability that my opponent will produce unit u exactly d intervals from now?" Importantly, such queries can be asked even when the values of some previous observation variables are unknown, for example due to limited scouting in an RTS game. As another example, HMMs can be applied to infer the most likely state sequence given an observation sequence. This allows us to infer the most likely strategy of a player based on observations, which can be useful for analysis and indexing purposes. Our experiments employ the above types of queries among others.

Learning

The model is learned from a set of training games. Each game is represented by a sequence of observation vectors $\vec{X} = \left[O^1, O^2, \ldots, O^T\right]$, where $O^t = (O_1^t, \ldots, O_U^t)$ is the binary observation vector of the production in interval t. The parameters of the HMM are learned using the Expectation Maximization (EM) algorithm (Dempster, Laird, and Rubin 1977; Rabiner 1990; Murphy 2002). EM is a local-search algorithm that maximizes the probability of the observations given the model parameters, $P(\vec{X}|\Phi)$. This quantity is known as the likelihood of the training sequence $\vec{X}$. The α and β parameters are initialized to $1/K$, and the θ parameters are initialized to random values drawn uniformly from the [0,1] interval. EM is iterated until convergence.

Experiments

We validate our approach by learning a model of the strategies of Protoss players in Protoss vs. Terran match-ups and assessing its utility. While our method can be applied to any playable race and match-up, providing reasonable discussion of all permutations is beyond the scope of this paper. We first describe how our data were collected, and give a qualitative analysis of the learned model and the discovered strategies. Then, we provide a quantitative analysis of the model's ability to predict future game states. Lastly, we use the model to identify unlikely sequences of states corresponding to novel strategies or erratic player behavior.

Data Collection and Model Selection

We collected 331 Protoss vs. Terran replays from the "Team Liquid"[1] and "Gosu Gamers"[2] websites. Both websites contain large archives of replays from expert players, including some South Korean professionals. The BWAPI library[3] was used to extract counts of the units owned by each player. For each game, the first 10,800 frames ($\sim$ 7min) of gameplay were extracted and divided into fourteen 720-frame ($\sim$ 30s) non-overlapping intervals. For each interval, the total number of units of each of 30 possible unit types produced by the Protoss player was counted and the counts collapsed into a vector of binary production values.

The main design decision in constructing an HMM is the choice of the number K of hidden states in the model. We compared several choices for K using five-fold cross-validation. In this process, the data is split into 5 non-overlapping blocks each containing 20% of the games. For each fold, 4 blocks are used for learning Φ, and the likelihood $P(\vec{X}|\Phi)$ is computed on the remaining held-out block. The likelihood is averaged over all five folds. In learning the model, we discarded the Probe and Pylon unit types, because they are produced in almost every time step and, hence, do not provide any useful information.

We evaluated models for $K = 18, 21, 24, 27$, and 30. We found no significant difference in likelihood across all sizes. The learned building preferences in the 30-state model best matched the intuition of our domain experts, so this model was selected. After model selection, all 331 games were used to fit the model parameters.

Model Analysis

Figure 2 depicts the state transition diagram learned by the model. Thicker edges correspond to higher transition probabilities, and all except a few edges with probability less than 0.25 have been removed for clarity. The labeled boxes surrounding groups of nodes represent our interpretations of the strategies embodied by the states inside each box.

States with a single, high-probability out-edge have high predictive power. Knowing that our opponent is in State 15, for example, is strong evidence that the next state will be State 14, and the one after that State 19. This sequence corresponds to the well-known *Reaver drop* strategy, [4] in which the Protoss player sacrifices early economic power to produce a powerful attacking unit and an airborne transport to carry it. The goal is to drop the Reaver off at the rear of our base and use it to destroy our workers. A successful Reaver drop can end the game in seconds by crippling our economy, but its success depends on surprise. If we believe that the opponent is in State 15, we can predict with high confidence, more than a minute in advance, that he intends to produce a Reaver. This advance knowledge is a tremendous advantage.

The model has learned two other high-probability sequences: the three states labeled "Early Game" and the four states labeled "Dark Templar." In the early game, there are

[1] http://www.teamliquid.net/replay/

[2] http://www.gosugamers.net/starcraft/replays/

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

[4] http://wiki.teamliquid.net/starcraft/1_Gate_Reaver

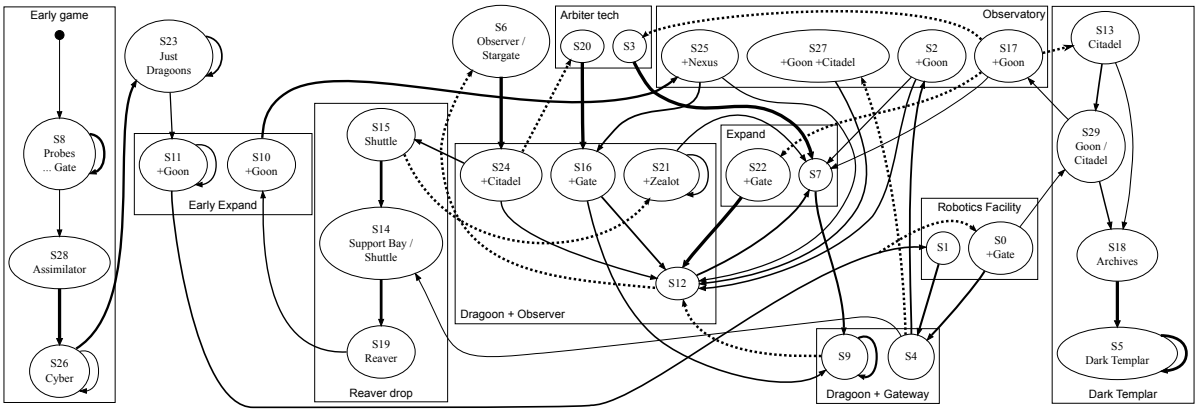

Figure 2: The state transition diagram learned by the model. Thicker edges denote larger transition probabilities (e.g., the edge from S20 to S16 has probability 1.0). The solid edges all have probability at least 0.25. Additional dotted edges (with probabilities of 0.05-0.15) are shown so that every node is reachable. State 8 is the initial state. The labeled boxes around groups of nodes are our interpretation of the strategy represented by those states. When a box is labeled with a unit type (such as "Observatory"), that unit type was likely to be produced in all states within the box.

few choices to make, and it is no surprise that most players in our training set built about the same units at about the same times. The "Dark Templar" cluster captures a second specialist strategy in which the goal is to attack with Dark Templar, a type of unit that is invisible to enemies unless a unit with the *Detector* ability is nearby. Like the Reaver drop, this strategy can end the game immediately if we are not prepared, but is easy to repel if we anticipate it.

The state diagram also features some states that have multiple out-edges with similar probability. In these states, we can narrow the Protoss player's next state down to a few possibilities, but have no reason to favor any one of them. This ambiguity indicates that it is a good time to send a scout to observe what our opponent is doing. Suppose that we believe that our Protoss opponent is in State 4. There is a high probability that his next state is either State 2 or State 14. In State 2, he will build an Observatory with probability nearly 1. In State 14, on the other hand, he will build a Robotics Support Bay with probability more than 0.9, but almost never an Observatory. Thus, if we send a scout during the next time interval and see an Observatory, we know that our opponent is most likely in State 2, pursuing a standard Observer opening. However, if our scout sees a Support Bay, we know our opponent is in State 14, going for a Reaver drop.

Prediction

As described earlier, prediction is handled in our model through probabilistic inference. We examine the results of two types of queries applied to a particular game involving a *Reaver drop* strategy. Query "A" is of the form $P(O_u^{t+d} = 1|O^0, O^1, \ldots, O^t)$. This query asks, "Given the observed production from times 0 to t, what is the probability that my opponent will produce unit type u exactly d intervals from now?" Query "B" is $P(O_u^{t:T} \neq 0|O^0, O^1, \ldots, O^{t-1})$ and asks "Given what I have seen up through $t-1$, what is the probability that my opponent will produce at least one unit of type u at any point in the future?"

Figure 3 shows the results of these queries for 2 different unit types—the Protoss Reaver and Observer. The barplots show the results of Query A after observing the game up to times $t = 5, 7, 9$, and 11, respectively. For example, barplot (1) has observations up to $t = 5$ (indicated by the black vertical line), and gives the build probabilities for Reavers and Observers in each time interval $t > 5$. The captions in each plot contain the result of Query B (labeled $P(\text{Reaver}^{t:13}|O^{0:t-1})$) asked at time t. The captions also give a running record of the observations made so far.

In (1), Query A with the 6 initial observations shows that a Reaver (green bar) is unlikely (< 0.05) to be made at any future time. However, when we see the Robotics Facility in (2), the probability of a future Reaver rises. The probability is still small because the Robotics Facility may indicate production of a much more common unit, the Observer (blue bar). In (2), we can interpret the green bar peaking at $t = 10$ as the model suggesting that, *if* a Reaver is built, it is most likely to be built 3 intervals from now. The delay is predicted because the transition model enforces that a Support Bay state must be visited before moving to a Reaver-producing state (e.g., the path $S1 \rightarrow S4 \rightarrow S14 \rightarrow S19$ in Figure 2). This query has thus successfully identified the time at which the Reaver was actually built as the most likely—1.5 minutes before its construction. Once we observe the Support Bay in (3), our confidence that a Reaver is coming in the next time interval jumps to 0.77 (near-certainty). When we finally see the Reaver constructed at $t = 10$ (4), our belief that another Reaver will be made by the end of the 7-minute game plummets. The model has learned that a self-transition to the Reaver production state is unlikely, which suggests that players rarely make two of this expensive unit. In (1), Query B tells us that, with little initial evidence, we expect the opponent to build a Reaver in about 24% of games. Once the Support Bay is seen (3), Query B matches Query A.

Observers (blue bar) are a commonly-produced unit in Protoss vs. Terran, which is evident from Query B in (1) yielding 0.64 probability at $t = 5$. Once we see the Robotics Facility (which produces both Reavers and Observers) in (2),

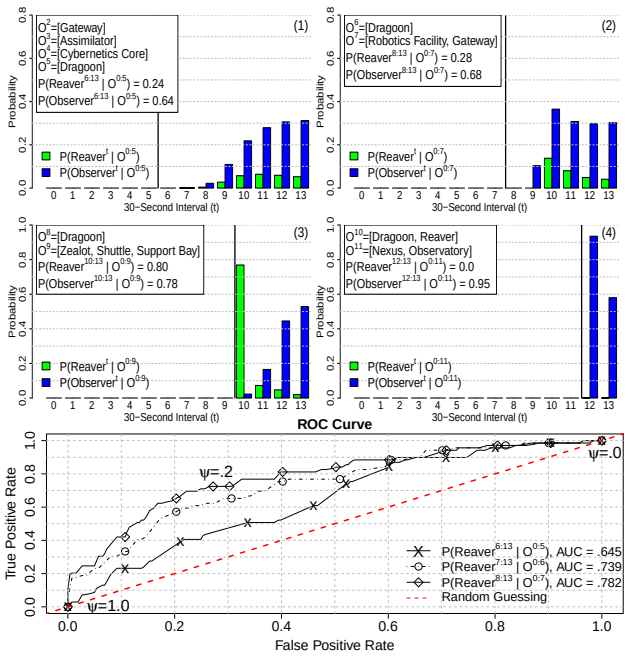

Figure 3: Top (Barplots): the prediction results for Query A applied to Reaver and Observer units for a single game. Bottom: the ROC curve for the Reaver-prediction task over all games (5-fold cross validation).

Query A expects an Observer three time-steps later at time 10. In this game, however, the Protoss player is pursuing a Reaver drop and will be delayed in building Observers. When the Reaver strategy becomes obvious after the Support Bay is built in (3), the probability of an Observer in the immediate horizon decreases. We cannot predict an Observer confidently until we see its precursor building, the Observatory, constructed at $t = 11$. After an Observer is built at $t = 12$ (not shown), we maintain significant belief (> 0.6) that another Observer will be built in the final time step. Unlike the Reaver, Protoss players often produce several Observers to spy on their opponents.

To assess the overall prediction accuracy of our models, we computed receiver operating characteristic (ROC) curves (Figure 3, bottom) for predicting (at times 5, 6, and 7) future production of a Reaver (Query B). If the predicted probability exceeds a threshold, ψ, we predict that a Reaver will be built in the future. The curve is created by varying ψ from 0.0 to 1.0. The horizontal axis (False Positive Rate; FPR) is the fraction of false positive predictions (i.e., fraction of times a Reaver was predicted when none was built); the vertical axis shows the True Positive Rate (TPR). FPR and TPR are computed from the 5-fold cross validation. The area under the ROC curve is equal to the probability that a randomly-chosen Reaver-containing game is ranked above a randomly-chosen Reaver-free game. The diagonal line corresponds to random guessing, and the area under it is 0.50.

Of the three type B queries given, the third query (based on evidence up to $t = 7$; diamond-line curve) performed the best. Using $\psi = 0.2$ as a threshold, a true positive rate of 0.725 was achieved while keeping a FPR of 0.272. Time

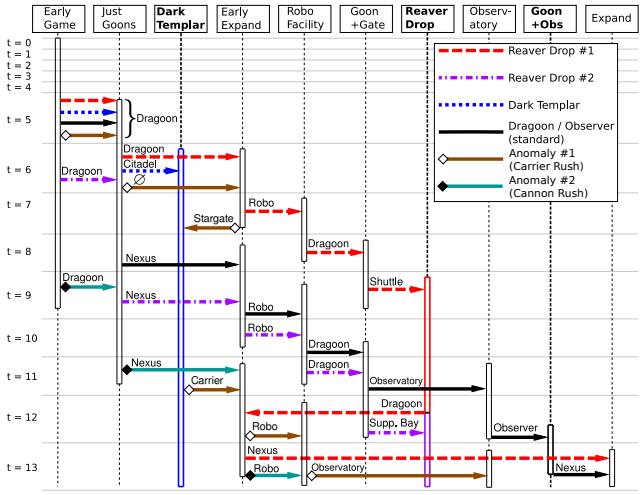

Figure 4: Cluster transitions for 4 typical games and 2 atypical ones. Each column represents a cluster identified in Figure 2. Edges represent transitions from one cluster to another, and are labeled with the unit observation that most likely triggered the transition. The undecorated arrows describe four games in which normal strategies were observed. The arrows with diamonds on their tails describe two of the games our model found most unlikely.

intervals 6 and 7 appear to be the earliest that Robotics Facilities are constructed in the games we saw, which explains why predictions made with evidence up to this point increase in accuracy. We consider this good performance given that the model only has knowledge for half of the game when making a prediction about a possible future Reaver.

Game Traces

The Viterbi algorithm (Rabiner 1990) can be applied to the learned model to compute the most likely sequence of states responsible for a given set of observations. We refer to such a sequence of states as a *game trace*.

We can interpret game traces as paths through the *clusters* in the state diagram (Figure 2). Clusters correspond to higher-level behaviors than the individual states, which allows us to examine the game at a higher level of abstraction. Figure 4 shows paths through the clusters for six different games. The solid arrows show a "standard" build order, in which the Protoss player takes an early expansion and then researches Observer technology. The dashed and irregular arrows show two different games in which the Protoss player attempted a Reaver drop. In the first game, the player went for a Reaver quickly and followed it up by taking an expansion, while in the second, the player took an expansion first and went for Reavers afterward. Despite the different temporal ordering of the build choices, the model detected the Reaver drop strategy in both cases before the Reaver was actually built. The trace shown with dotted arrows was a Dark Templar game. This trace illustrates a weakness of our model. Although the Protoss player actually built Dark Templar for only a single time step before proceeding to take an expansion, the high self-transition probability of the Dark Templar state (State 5) outweighed the influence of the observations, causing the model to predict more Dark Templar.

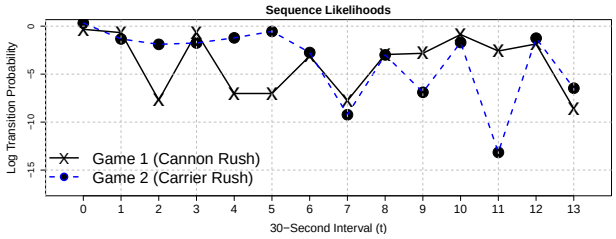

Figure 5: The relative likelihood of transitioning from state $t-1$ to t for the duration of two unusual games.

Identifying Unusual Games The Viterbi algorithm also returns the overall likelihood of the best path given the parameters of the HMM. We can find unlikely games by examining these likelihoods. We can then calculate the likelihood of transitioning through the states in the game traces in order to determine what parts of each game our model finds unlikely. The transition likelihood is given by

$$\ln\left(\frac{P(S_t = k|S_{t-1} = j, O^{1:t})}{P(S_{t-1} = j|S_{t-2} = i, O^{1:t-1})}\right),$$

where k, j, and i correspond to the most likely states at times $t, t-1$, and $t-2$ in the game trace. Large negative values indicate unlikely transitions from the previous time interval.

We examined the five least-likely games in our dataset. Generally, we found that they featured strategies that would be risky or ineffective against skilled opponents. The likelihood traces for two of these games are shown in Figure 5. Game 1 demonstrates a *Cannon rush* strategy, in which the Protoss player uses defensive structures (Photon Cannons) offensively by building them in his opponent's base. The Cannons are defenseless while under construction, so the rush will fail if the opponent finds them in time. This strategy is rare in high-level play because it will almost always be scouted. The model gives low likelihood to intervals 2, 4, 5, and 7, when the player constructs a Forge, Cannon, Cannon, and third Cannon. From the game trace (Figure 4; filled diamonds), we see that the most likely state sequence did not leave the "Early Game" cluster until much later than the more typical games, since the Protoss player was spending money on Cannons rather than on early development.

Game 2 shows a very unusual *Carrier rush* strategy. Carriers are an advanced Protoss technology, typically seen only in the late game. To build one in the first seven minutes, the Protoss player must limit investment in military units, making this strategy tremendously risky. It is a "fun" strategy, which one will not see in high-level play. The deepest dips in likelihood (Figure 5) correspond to the decisions to build a Stargate ($t = 7$), Fleet Beacon ($t = 9$), and Carrier ($t = 11$), as shown in the game trace (Figure 4; open diamonds).

Summary and Future Work

This work investigated a probabilistic framework, based on hidden Markov models, for learning and reasoning about strategic behavior in RTS games. We demonstrated our approach by learning behavior models from 331 expert level *Starcraft* games. The learned models were shown to have utility for several tasks including predicting opponent behavior, identifying common strategic states and decision points, inferring the likely strategic state sequence of a player, and identifying unusual or novel strategies. We plan to extend this initial investigation in several directions. First, we are interested in incorporating partial observability into the model, and using the learned behavior models to optimize scouting activity. In particular, scouting should be directed so as to acquire the observations most useful for reducing uncertainty about the opponent's strategy. Second, this work has used a relatively simple model of behavior, both in terms of the observations considered and the transition model. We plan to extend this by allowing states to encode more refined information about production rate and by using transition models that explicitly represent state duration. Third, we are interested in extending the model to account for activity throughout full games rather than just the first 7 minutes. This will include inferring behavior states related to tactical activities such as attacking and defending. Fourth, we are interested in demonstrating that such predictive models can be used effectively in Monte-Carlo planning for RTS game AI.

Acknowledgements

This research was partly funded by ARO grant W911NF-08-1-0242. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO or the United States Government.

Jesse Hostetler is supported in part by a scholarship from the ARCS Foundation, Portland, OR.

References

Aha, D. W.; Molineaux, M.; and Ponsen, M. 2005. Learning to win: Case-based plan selection in a real-time strategy game. *Case-Based Reasoning Res. Dev.* 5–20.

Balla, R., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *IJCAI*, 40–45.

Chung, M.; Buro, M.; and Schaeffer, J. 2005. Monte Carlo planning in RTS games. In *IEEE CIG*, 117–124.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *JRSS B* 39(1):1–38.

Hsieh, J., and Sun, C. 2008. Building a player strategy model by analyzing replays of real-time strategy games. In *IJCNN*, 3106–3111. IEEE.

Murphy, K. 2002. *Dynamic Bayesian Networks: Representation, Inference, and Learning*. Ph.D. Dissertation, University of California, Berkeley, Berkeley, California.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2007. Case-based planning and execution for real-time strategy games. *Case-Based Reasoning Res. Dev.* 164–178.

Rabiner, L. R. 1990. A tutorial on hidden Markov models and selected applications in speech recognition. In *Readings in speech recognition*. Morgan Kaufmann. 267–296.

Schadd, F.; Bakkes, S.; and Spronck, P. 2007. Opponent modeling in real-time strategy games. In *8th Int'l Conf. on Intelligent Games and Simulation*, 61–68.

Weber, B., and Mateas, M. 2009. A data mining approach to strategy prediction. In *IEEE CIG*, 140–147. IEEE.