Identifying Players and Predicting Actions from RTS Game Replays

Christopher Ballinger, Siming Liu and Sushil J. Louis

***Abstract*— This paper investigates the problem of identifying a Real-Time Strategy game player and predicting what a player will do next based on previous actions in the game. Being able to recognize a player's playing style and their strategies helps us learn the strengths and weaknesses of a specific player, devise counter-strategies that beat the player, and eventually helps us to build better game AI. We use machine learning algorithms in the *WEKA* toolkit to learn how to identify a *StarCraft II* player and predict the next move of the player from features extracted from game replays. However, *StarCraft II* does not have an interface that lets us get all game state information like the BWAPI provides for *StarCraft: Broodwar*, so the features we can use are limited. Our results reveal that using a J48 decision tree correctly identifies a specific player out of forty-one players 75% of the time. This decision tree can identify a player within the first nine minutes of the game. Finally, we can correctly predict what action out of fifty possible actions a player will choose to do next 81% of the time. For a player we have already identified, we can correctly predict the action 89% of the time. This work informs our research on identifying and learning from Real-Time Strategy players and building better game AI.**

I. Introduction

THE Real-Time Strategy (RTS) game genre has become a popular focus of attention for Artificial Intelligence (AI) research in recent years [1]. RTS games are a type of strategy video game where players need to gather resources, build structures, research technologies and conduct simulated warfare. Understanding each of these factors and their impact on decision making is critical for winning an RTS game. One of the key challenges for RTS game designers is creating AI opponents which are able to adapt their strategies to different player actions. This requires AI opponents capable of identifying a player's strategy. Therefore, learning to recognize a player from the way the player plays the game could help us to build a player model, which we can use to devise counter-strategies that beat the player. Furthermore, if the player plays exceptionally well, we can learn strong winning playing styles and strategies from analyzing the player's decision making process. In this paper, we focus on using machine learning techniques to identify professional *StarCraft II* players from a database of game replays. We expect this study to help us understand decision making in RTS gameplay and therefore build better artificial RTS game players.

Christopher Ballinger, Siming Liu and Sushil J. Louis are with the Department of Computer Science and Engineering, University of Nevada, Reno. 1664 N. Virginia Street, Reno Nevada, United States (email: {caballinger, simingl, sushil}@cse.unr.edu).

This research was supported by ONR grants N00014-12-1-0860 and N00014-12-C-0522.

StarCraft II, seen in Figure 1, is one of the most popular multi-player RTS games with professional player leagues and world wide tournaments [2]. In South Korea, there are thirty two professional *StarCraft II* teams with top players making six-figure salaries and the average professional gamer making more than the average Korean [3]. *StarCraft II*'s popularity makes obtaining larges amounts of different combat scenarios for our research easier, as the players themselves create large databases of professional and amateur replays and make them freely available on the Internet [4]. However, unlike *StarCraft: Broodwar*, which has the *StarCraft: Brood War* API (BWAPI) to provide detailed game state information during a live game, *StarCraft II* has no such API [5]. This means that the information we have access to is limited to the player actions recorded in the replay files.

Fig. 1. Snapshot of StarCraft II.

Board games and card games have a long history of being used for AI research, dating back as far as the early fifties with Samuel's work on developing a checkers player[6]. However, RTS games provide a bigger and more challenging problem space. RTS games involve resource management, spatial reasoning, strategic thinking, and tactical thinking. A player has to build up an economy to obtain enough resources to generate and support a strong military that will defeat their opponent. Players also must adapt to only having partial game state knowledge. In *StarCraft II*, a "Fog of War" obscures areas of the map where a player does not have any units nearby. Players must send some units to search an area, or "scout", to learn where their opponent is located and what their opponent is building, so the player can adjust their plans

accordingly. Any advances in AI approaches to learning from RTS game players will have industrial, military, and social applications.

A typical *StarCraft II* match takes place on a $width \times height$ sized map which is measured in the number of 32×32 pixel squares, also known as build tiles. The typical map size ranges from 64×64 to 256×256 build tiles. Each player controls an unlimited number of buildings, in addition to many different units with a combined "Supply Cost" up to 200. Every unit in *StarCraft II* costs a certain number of supplies to construct, and each type of unit has a different cost. For example, a worker unit costs 1 supply to build, while a military unit called a "Battleship" costs 8 supplies. The supply cap is 200 by default in every one-versus-one match. From a theoretical point of view, the state space of *StarCraft II* for a given map is enormous and it is almost impossible to do a tree search for an optimal game state. For example, considering the location of each unit (with 64×64 possible positions per unit), 400 units gives us $(64 \times 64)^{400} \approx 8.8 \times 10^{1444}$. Therefore tree search techniques which are widely applied on board games such as chess are not applicable on RTS games [7].

Our overall research goal is to develop competent RTS game players, and this paper informs our research in this direction. We analyze game replays from professional and amateur *StarCraft II* players, in order to see if we can identify which player is in a game and predict what action they will take next. If we can identify a player, how much time must pass in the game before we can make a prediction? Can we identify the strategy that the player is using? Can we predict the players next move based on the previous actions the player performed? In our research, we explore the use of supervised machine learning techniques on *StarCraft II* game replays to address these questions. Specifically, we apply machine learning algorithms from the *WEKA* toolkit to features extracted from *StarCraft II* replay files, in order to learn to identify forty-one *StarCraft II* players and predict the next action they will take at different parts of the game. Our results show we can correctly identify a specific player out of forty-one players 65% of the time, using mostly features from the first nine minutes of a game. We also show that we can predict actions out of fifty possible actions an unknown player will choose to do next 81% of the time, while we can correctly predict actions 89% of the time for a player we identified beforehand.

The rest of this paper is organized as follows: Section II describes related work in RTS games and in player modeling. The next section introduces our methodology and features used for player identification and action prediction. Section IV presents preliminary results and discussion. Finally, section V draws conclusions and discusses future work.

II. Related Work

StarCraft II was released in 2010 and, being a relatively new game, has not been used much for scientific research. Michael Whidby implemented a Python game for studying scouting efficiency in different leagues from one-versus-one games in *StarCraft II* [8]. His results, for a specific kind of scouting, shows that players in higher leagues scout more than players in lower leagues.

However, *StarCraft: Brood War*, the predecessor to *StarCraft II*, has been used often for research in the AI community. Ji-Lung Hsieh and Chuen-Tsai Sun applied a case-based reasoning approach for the purpose of training their system to learn and predict player strategies [9]. Ben G. Weber and Michael Mateas presented a data mining approach to opponent modeling in *StarCraft* [10]. They applied various machine learning algorithms to detecting an opponent's strategy in game replays. Then they used the learned models to predict the opponent's strategic actions and timing. If you can predict what your opponent is doing, it is usually fairly straightforward to find a good counter strategy and defeat your opponent. Note that this is the main reason that industrial game AI for RTS games is easy to beat. Predictability is often a fatal weakness in these games.

Player identification research often uses other types of games as research platforms. Jan Van Looy and Cedric Courtois studied player identification in online games [11]. They were focused on massively multiplayer online games (MMOGs). Their research did not use game state information, rather, their research was based on a group of volunteers who gathered data on the preferences of their avatar's appearance using survey questions. Josh McCoy et al. present a RTS game AI architecture that integrates multiple specialist components to play a complete game in the RTS game *Wargus* [12]. Bakkes et al. worked on a novel approach that enables the game AI to gather domain knowledge automatically and immediately utilize the knowledge to evoke effective behavior in the RTS game *SPRING* [13].

Hladky Stephen et al. used hidden semi-Markov models and particle filters as a means for predicting opponent positions in the first person shooter game *Counter-Strike: Source* [14]. Their models can perform with similar or better accuracy than the average human expert in this game. Alex Fink et al. applied a general approach to learn the behavior of non-player characters (NPCs) and other entities in a game from observing just the graphical output of the game during game play in the video game *Pong* [15]. They use object tracking and situation-action pairs with the nearest-neighbor rule, and correctly predict the next behavior of the computer controlled opponents 9 out of 10 times.

Bayesian networks have been used for strategy recognition in games as well. Charniak et al. proposed a method where they first retrieve candidate explanations and assemble these explanations into a plan recognition Bayesian network [16]. Then they perform Bayesian updating to choose the most likely interpretation for the set of observed actions. David W. Albrecht presented an approach to keyhole plan recognition which uses a Bayesian network to represent features of the domain that are needed to identify users' plans and goals in a Multi-User Dungeon adventure game [17].

Some work has been done in extracting features from replay files. *SC2Gears* provides a *StarCraft II* replay parsing

service to convert a binary replay file to an XML structured file which we can easily understand [4]. Gabriel Synnaeve and Pierre Bessiere worked on extracting the complete game state from a recorded *StarCraft: Broodwar* replay file by rerunning the replay file and recording the complete game state through the BWAPI. This approach enables access to the complete game state for every frame in the game. However, *StarCraft II* does not have such an interface yet so we cannot access its complete game state. We therefore only use the data from player actions in *StarCraft II* replay files as parsed by the *SC2Gear* parsing service. To the best of our knowledge, this is the first work to use data from *StarCraft II* to identify players and predict player actions.

III. Methodology

The first step of this research was collecting enough game replays from several specific players. A *StarCraft II* replay file stores all user action events in a game match. These lists of user-actions reflect the players' thinking and decision making during a game, and we therefore believe that we can infer the players' playing style and find useful strategic patterns from replays.

A. Data Collection

There are many websites dedicated to collecting and sharing user and team contributed game replays. Many of these replays are from pro-leagues and professional tournaments such as *Major League Gaming, IGN Pro League, Global StarCraft II League*, and *StarCraft II Pro League* [18], [19], [20], [21]. Therefore, we are able to collect a representative set of replays for specific professional players. We only focus on the one-versus-one type of match because it is the most popular game type for professional matches. Three different races (Terran, Protoss, and Zerg) are available for players to choose from at the beginning of each match. Each race is significantly different from others in terms of units, structures, technologies, and playing styles. In our previous paper, we limited our focus to Protoss players versus other races and got high accuracy for identifying a single player. In this paper, we cover all three races against other races.

We collected 3671 replays from *SC2Rep.com* and *GameReplays.org* [22], [23]. This dataset covered 41 different players against other random players. Since all the replays are uploaded by *StarCraft II* fans, there is no validation during their upload that the players in the replay are who they say they are, so the data can be noisy. For example, a Terran versus Protoss match may be mislabeled as Terran versus Zerg, or Player A may be mislabeled as Player B. We had to clean up the noisy data manually, and finally ended up with 2110 replays. The dataset is still noisy after the clean up, but it is much better than the original dataset. The breakdown of these games among the races is shown in Table I.

StarCraft II replay files are stored in a binary format and record all player actions during the game. The data in the file is not stored in a human readable format. We need to parse the replay file to a format that we can understand and use. Several websites provide a *StarCraft II* replay parsing service. We used *SC2Gears* to convert raw replays to XML structured files. Therefore, we get all player performed actions from the parsed replay files and know exactly what players did in the match. Note that when the replay file is used by the game engine to replay the match, all game states must be recalculated in real time using the user input for all players stored in the replay file, the replay file doesn't contain any game state information. This limits our knowledge extraction from replay files. A player's current units, structures, and resources at a particular time in the game are not available in replay files. In contrast, BWAPI for *StarCraft: Broodwar* could get complete game state information by running replay files and recording the game states with user plug-ins, but *StarCraft II* does not have this interface yet. Table II shows a sample of an arbitrarily selected parsed game replay of a Protoss player and a Terran opponent.

TABLE I
Game Distribution in Replay Files

Replay Types	Number of Replays
Terran versus Terran	281
Terran versus Protoss	453
Terran versus Zerg	498
Protoss versus Protoss	208
Protoss versus Zerg	300
Zerg versus Zerg	370
Total replays	2110

TABLE II
Replay Logs

Frame(Time)	Player	Action	Object
3296	Player 1	Build	Pylon
3588	Player 2	Build	Supply Depot
3625	Player 1	Train	Probe
4804	Player 2	Train	SCV
5638	Player 1	Select	Hotkey 1
6208	Player 2	Build	Barracks
7543	Player 1	Attack	Target position

B. Feature Extraction

The goal of our feature extraction is to maximize the capture of game information and aspects that expose player-specific traits, so we can identify each player from other players based on their unique characteristics. We create and use a feature vector composed of three parts. The first part is general game information which includes game length, winner, and actions per minute (APM). The second part represents the changing state of the game and covers how many units or structures are built in three minute time slices. The last part records the time (as a frame number) for the first build or use of each building, unit, and unit ability.

Formally, our features are represented by Equation 1 where x is units, structures, upgrades, or abilities, and t is the index

TABLE III
SAMPLE BUILD ORDER

Player 1 (Zerg)	Player 2 (Terran)
Train Drone	Train SCV
Train Drone	Train SCV
Train Overlord	Train SCV
Build Spawning Pool	Build Supply Depot
Train Drone	Build Barracks
Train Drone	Train SCV
Train Zergling	Train Marine

of a three minutes time slice with $t \leq 10$.

$$\overrightarrow{F} = \{G, S_x^t, O_x\} \quad (1)$$

Here, $\overrightarrow{F}$, the feature vector is made up of three parts. First, G represents general game information and contains game length, winner, map name, game version, etc. Second, S_x^t represents the number of x produced in time slice t. And third, O_x is given by Equation 2.

$$O_x = \left\{ \begin{array}{ll} f, & \text{frame that } x \text{ was first produced} \\ 0, & \text{if } x \text{ was never produced} \end{array} \right\} \quad (2)$$

In the end, we collected 230 features from each replay file.

C. Strategy Extraction

In addition to player identification, we are also interested in strategy recognition. However, the features extracted with the previous time-based approach is more related to players' playing statistics. Generally speaking, a strategy used by a player is how the player prioritizes building structures, training units, researching technologies, and launching attacks. A player's build-order contains most of the above information, especially at the early stage of the game. For example, the *6 Pool Rush* strategy used by Zerg players is building no workers at all at the beginning of the game, and instead saving resources to build *Spawning Pool* to launch an attack as soon as possible. For a *Fast Expansion* strategy, the player will build large numbers of workers early to maximize their economy as fast as possible, and build more military units at the middle or late stages of the game. In our research, we extract build-order as our second feature vector and use it to represent a strategy.

D. Evaluation

Various classification and prediction techniques in the *WEKA* toolkit from the *University of Waikato* were applied to identify a player out of forty-one players [24]. *WEKA* is a powerful machine learning software that includes several classification and prediction algorithms, as well as preprocessing and regression techniques and tools for statistical analysis. In our previous study we applied the following techniques:

- J48 - C4.5 Decision Tree
- ANN - Artificial Neural Networks
- AdaBoost - Adaptive Boosting
- Random Forest - Ensemble classifier that consists of many decision trees

J48 and ANN use default parameters provided by *WEKA*. AdaBoost was configured to use the J48 decision tree with default settings. Random Forest was configured to use 10 random trees with nine random features. In this study, we use primarily J48 and Random Forests since those methods had very similar results to ANN and AdaBoost in our previous study, and because our large feature vector in this study did not always work with ANN and AdaBoost.

IV. RESULTS AND DISCUSSION

Several machine learning techniques were applied to get high identification and prediction performance with the WEKA toolkit. We used ten fold cross validation during the training and all accuracy results are on the testing set. In our previous paper, we identified whether or not a particular Protoss player was in a game. Our identification performance was as high as 87.6% by using AdaBoost in all PVP games, which revealed that we can reliably and precisely identify a professional player among other professional players based only actions during the game. Figure 2 from our previous work compares the identification accuracies of using all attributes for each algorithm and race. The results also indicate that a professional gamer seems to have a unique playing style. The features we extracted from replays kept a player's unique characteristics.

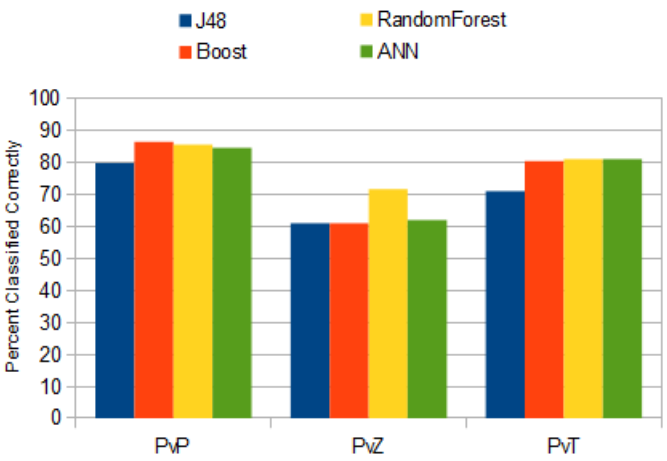

Fig. 2. Accuracy of identifying our Target Player using time-based attributes from our previous paper.

The decision tree algorithm used by J48 is not only fast, but allows easy conversion from a tree to a set of rules; rules which may be better understood by people. On the other hand, the representations used by ANNs, AdaBoosted trees, and Random Forests are harder to understand. We thus analyzed rules generated by high performing decision trees generated by J48. Below is an example of a single rule from PVP games generated by the decision tree. If a game's

features matched this rule, then we know it is our target Protoss player, otherwise, it is not.

- Worker count built in first 3 minutes greater than 24, and
- First Gateway was built before frame 6320, and
- Blink was used less than 12 times between 6 to 9 minutes, and
- First Chrono Boost was used after frame 4692.

In this paper, we identify which player out of forty-one possible players is in a game, and predict what actions they will take at different points in the game.

A. *Performance of Player Identification Out of Forty-one*

In our current experiments, we extend our player identification focus from a specific Protoss player to 41 different players. All the replays covered both professional players and high ranked amateur players. Figure 3 shows the average accuracy of identify a player out of 41 players, both with APM and without APM. This shows that APM is a strong indicator of the player, and allows us to correctly identify the player 75% of the time out of 41 possible players with Boosted J48. Considering a player we are trying to identify in a game could be incorrectly identified as any of the 41 different players in out dataset, 75% is surprisingly good. Without APM and basing our identifications only on player actions, we can correctly identify players 43% of the time, which is still much better than the probability of guessing at random. This proves that a professional players APM is relatively consistent in tournaments and that APM is a strong feature for identifying a player.

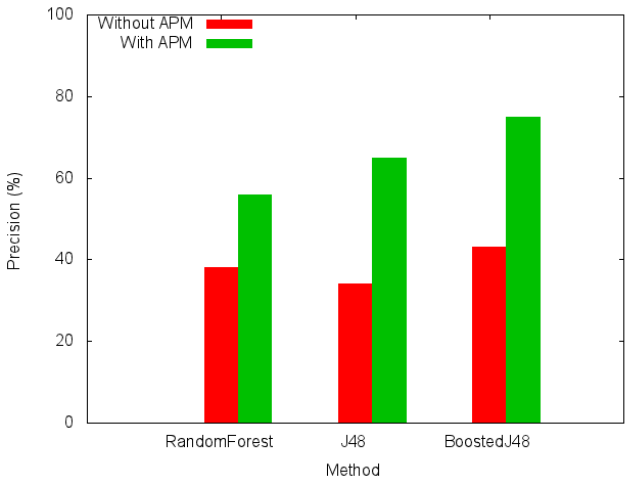

Fig. 3. Precision of identifying our Target Player out of 41 players using time-based attributes.

The results also show that when a player is misidentified, most misidentifications are allocated to two or three different players, and not randomly spread among all players. This may indicate that all these players share similar or common strategies and playing style. For example, two thirds of misclassification on Player T are misclassified as Player P and Player F. This is because Player T, Player P, and Player F are known to have a similar playing style.

We are also interested in how a specific player performed in our experiments. We chose a player in our dataset to be our Target player. For this specific player, our identification accuracy is 75% with APM and 44% without. Figure 4 shows the performance of identifying the Target player with time based attributes and build-order attributes without APM. It also compares the performance of identifying the Target player with the average performance of all players.

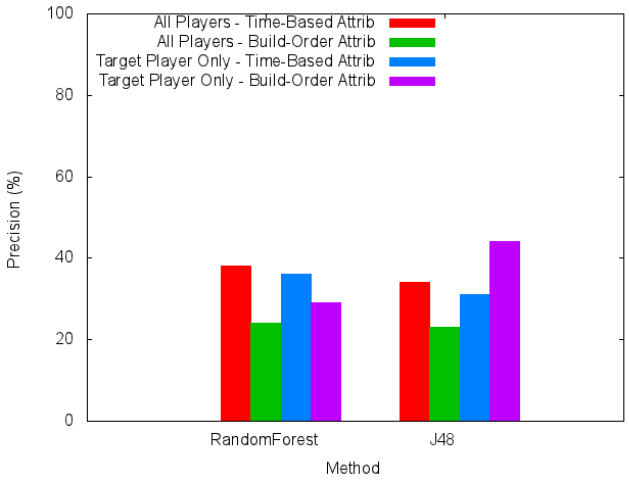

Fig. 4. Precision of identifying players out of 41 players based on different sets of attributes.

From Figure 4, we can see that in general the prediction accuracy based on time-based attributes is better than the prediction accuracy on build-order attributes. This is because time-based attributes generalize strategies more than the build-order attributes. For example, how many unit X were built in first three minutes? How many structure Y were built 6-9 minutes into the game? The build sequence does not matter in time-based attributes. However, build-order attributes are exactly what a player built in the game, and the order the units were built shows the different strategy used in the game. Therefore, the prediction on time-based attributes (first column) is much better than the prediction on build-order attributes (second column). However, the accuracy of identifying our Target player based on time-based attributes is worse than the accuracy based on build-order attributes. build-order attributes precisely present the order players build their units. If a player consistently builds units in precisely the same order, the build-order will becomes a strong indicator to identify a player. For example, some players are famous for using the *Two Bases All In* strategy. In which, the player can safely transit to a second base, build a huge amount of units, and launch the all in attack while his opponent is not strong enough to defend. In such a case, a consistent build-order is a strong indicator for a specific player. The results in Figure 4 showed that the prediction performance on build-order attributes of our Target player is better than generalized time-based attributes. However,

for players whose playing style is versatile, and do not build units in precisely the same order for the same overall strategy, the outcome might be different. There are several professional players whose playing styles are well known as being inconsistent, making it hard for other players to predict what they will do.

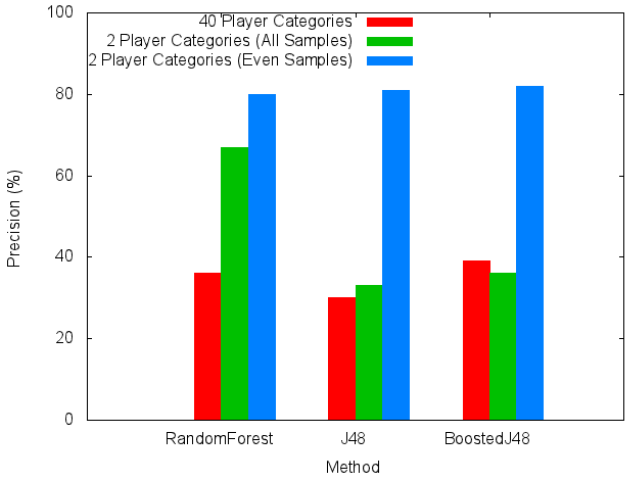

Fig. 5. Precision of identifying Target player out of different groups of players using time-based attributes.

We also worked on identifying our Target player out of different groups of players using time-based attributes. Figure 5 shows the accuracy of identifying our Target player out of different groups of players. First we try to identify our Target player with an even number of samples, meaning we have 155 positive cases which were played by our Target player and 155 negative cases classified as "NotTarget Player" taken from a random sample of all other players. All three techniques performed almost the same. Random Forest accuracy is 80%, J48 accuracy is 81%, and Boosted J48 accuracy is 82%. However, we also try identifying our Target player based on all samples, which contains 155 positive cases and 1955 negative cases. The green bars shows the results of this on all three techniques. The Random Forest accuracy is 67%, J48 accuracy is 36%, and Boosted J48 accuracy is 39%. J48 and Boosted J48 do worse than random, while the Random Forest does a little better than random. The reason is that the number of samples is so unbalanced that by constantly identifying the player as a non-Target player, we will get accuracy as high as $(2110 - 155)/2110 = 92.7\%$. This indicates that we should not use unbalanced sample to predict a specific target player. However, the first column shows that the average accuracy of player identification on all 41 player categories is around 36%, the Boosted J48 gets the highest accuracy of 39%. Which is surprisingly good for predicting one player out of 41 possibilities.

B. Action Prediction

In addition to identifying players, we want to predict what a player will do in the game. We extract exact build-orders from replays, train our model with a list of actions that the player already performed, and predict what the player will build at different points in the game. Figure 6 shows the correctly predicted actions using our build-order attributes. We can see that our accuracy for predicting action 50, which is on early of the game, is 59%. As the game goes on, the accuracy on predicting action 150 increased to 68% accuracy. On late game predictions, the accuracy of predicting action 250 climbed to 81% accuracy. This means the strategy is not consistent as consistent earlier in the game. Players may change their build-order based on their opponent's strategy. However, if a player chose a strategy on early game, it is less likely to change to a different strategy later on. Later in the game, their strategy is more consistent and it is easier to predict.

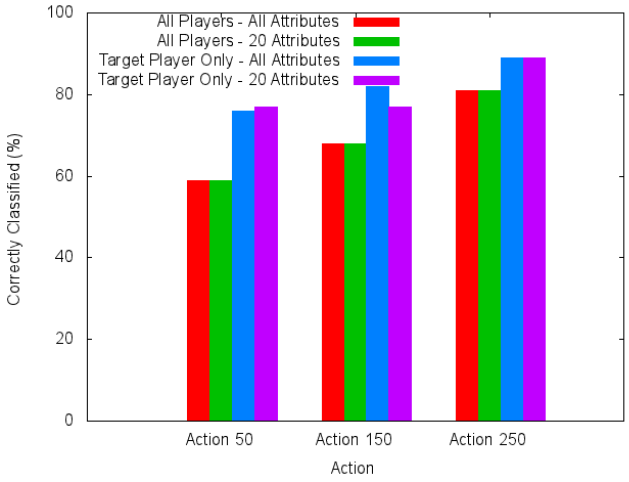

Fig. 6. Correctly predicted actions using build-order attributes.

We looked into the mis-predicted actions and one interesting finding is that our mis-predicted actions are usually predictions of a worker unit being built. This is because at the early stage of the game, most players heavily focus on training workers to maximize their economy for middle or late games. For the middle and late stages of the game, professional players still needs to train workers. In *StarCraft II*, destroying your opponents economy is one of the most important winning tactics, and professional players are good at harassment by attacking their opponents workers. Therefore, training workers can occur at any point in the game as a response the opponents harassment tactics, and not the players planned strategy. Our mis-predictions then usually mis-predict worker instead of the correct unit.

Figure 6 also shows that if we have already identified a player and try to predict the next move, our accuracy is much higher than predicting moves based on all players in general. We correctly predicted action 50 of our Target player 77% of the time, while we only make correct predictions 59% on all players. We got 82% accuracy on action 150, and 89% on action 250. We are also interested in optimizing our methods. For a certain action, more recent previous actions are much more important than actions performed long ago. If we can train our model with less data, it might improve

the speed to get the same accuracy. We then trained only previous 20 attributes on our model instead of all previous actions. The results in Figure 6 shows that the prediction on only 20 previous moves is the same accuracy as all attributes. This results could save memory resources and computational time to get the same accuracy as applying all previous moves. These results also show the benefit quickly identifying a player has when we need to predict the players strategy and create a counter-strategy.

V. Conclusions

Our research focuses on applying machine learning techniques to identifying Real-Time Strategy players and predicting their actions from game replay history. Our dataset consists of 2110 *StarCraft II* replays including 41 professional and high rank amateur players. We then used two types of feature extraction approaches and extracted two feature vectors. The first one is a time generalized feature vector which includes general game information, units and structures that are built in each three minutes time slice, and the time snapshot of when each type of units and structures were first built. The second feature vector includes only player build-orders, which precisely represent strategies of players used in the games. With these two types of feature vectors, we attempt to identify which player out of 41 possible players is in each game replay. We are also interested in recognizing the strategy a player used in a game. We use build-orders to represent a strategy and tried to predict the next action the player will build at different times in the game based on his previous actions.

In our previous study, we identified whether or not a known professional player participated in a game or not. Our results showed that we could correctly identify if our player was in a game 87.6% of the time. The J48 tree that was produced showed rules that differentiated our selected player from most other players based on the actions the players took. From these rules, we could see basic strategies the player uses, which betters our understanding of how he plays and helps us to find counter-strategies that disrupt the players usual style of playing.

Our current results show that without looking at the player's Actions Per Minute, we can identify a player out of 41 possible players with an accuracy up to 43% using AdaBoost and J48. If we take the players APM into account, the identification accuracy was as high as 75%. This reveals that we can reliably identify a professional player among many other player based on actions taken during a game. We also see that APM is a strong characteristic for indicating a player's identity. In both cases, samples that are misidentified are mostly misidentified as one or two different players out of all 41 possible players. This indicates that the correct player identity and the identities the sample was misclassified as may belong to players with very similar strategies.

We then use the feature vector which contains only build-orders to predict the next action the player will take based on their previous actions. Our results show that 59% of the time we can correctly predict what unit will be built at the early stage of the game, 68% of the time at the middle stage of the game, and 81% of the time later in the game. Our results also show that the accuracy goes up if we correctly identify or narrow down the possibilities of who the current player is. Similar to our player identification results, most mis-predictions were caused by a single source. Often instead of predicting the correct unit, a worker unit would be predicted instead. This is due to players responding to having their worker units destroyed by their opponent's strategy, instead of being apart of the players planned strategy.

In our future work, we are interested in automatically classifying different "types" of strategies. Build-orders that accomplish that same goals, but slightly change the order of a couple units can be considered the same type of strategy. Strategies that attempt to accomplish the same goals in a similar manner can often be defeated using the same counter-strategy. Furthermore, we want to predict what type of strategy a player, rather than predict all their actions.

References

[1] M. Buro, "Real-time strategy games: A new ai research challenge," *Proceedings of the 18th International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence*, pp. 1534–1535, 2003.

[2] Blizzard. (2014) Blizzard entertainment. [Online]. Available: http://www.starcraft2.com

[3] T. Liquid. (2014) Category: Korean teams. [Online]. Available: http://wiki.teamliquid.net/starcraft2/Category:Korean_Teams

[4] A. Belicza. (2012) The SC2Gear website. [Online]. Available: https://sites.google. com/site/sc2gears/

[5] M. Buro and D. Churchill, "Real-time strategy game competitions." *AI Magazine*, vol. 33, no. 3, pp. 106–108, 2012.

[6] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 44, no. 1.2, pp. 206 –226, jan. 2000.

[7] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," 2013.

[8] M. Whidby and S. C.-B. Bederson, "Sacovie-zerg scouting game for starcraft ii," *CHI, may*, 2012.

[9] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, june 2008, pp. 3106 –3111.

[10] B. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, sept. 2009, pp. 140 –147.

[11] J. Van Looy, C. Courtois, M. De Vocht, and L. De Marez, "Player identification in online games: Validation of a scale for measuring identification in mmogs," *Media Psychology*, vol. 15, no. 2, pp. 197–221, 2012.

[12] J. McCoy and M. Mateas, "An integrated agent for playing real-time strategy games," in *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, ser. AAAI'08. AAAI Press, 2008, pp. 1313–1318. [Online]. Available: http://dl.acm.org/citation.cfm?id=1620270.1620278

[13] S. Bakkes, P. Spronck, and J. van den Herik, "Rapid adaptation of video game ai," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 79–86.

[14] S. Hladky and V. Bulitko, "An evaluation of models for predicting opponent positions in first-person shooter video games," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 39–46.

[15] A. Fink, J. Denzinger, and J. Aycock, "Extracting npc behavior from computer games using computer vision and machine learning techniques," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*. IEEE, 2007, pp. 24–31.

[16] E. Charniak and R. P. Goldman, "A bayesian model of plan recognition," *Artificial Intelligence*, vol. 64, no. 1, pp. 53–79, 1993.

[17] D. W. Albrecht, I. Zukerman, and A. E. Nicholson, “Bayesian models for keyhole plan recognition in an adventure game,” *User modeling and user-adapted interaction*, vol. 8, no. 1-2, pp. 5–47, 1998.
[18] I. Major League Gaming. (2014) Major league gaming. [Online]. Available: http://www.majorleaguegaming.com
[19] I. IGN Entertainment. (2014) Ign entertainment. [Online]. Available: http://www.ign.com/ipl
[20] GOMTV. (2014) Global starcraft ii league. [Online]. Available: http://www.gomtv.net
[21] KeSPA. (2014) Starcraft ii proleague. [Online]. Available: http://e-sports.or.kr/proleague2014/
[22] (2010) The SC2Rep website. [Online]. Available: http://www.sc2rep.com/
[23] (2012) The GameReplays website. [Online]. Available: http://www.gamereplays.org /starcraft2/replays.php?game=33
[24] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278