Learning to Identify Rush Strategies in StarCraft

Teguh Budianto, Hyunwoo Oh, Takehito Utsuro

Learning to Identify Rush Strategies in StarCraft

Teguh Budianto[1], Hyunwoo Oh[2], and Takehito Utsuro[1]

[1] Grad. Sch. of Systems and Information Engineering, University of Tsukuba, Japan
[2] Grad. Sch. of Interdisciplinary Information Studies, The University of Tokyo, Japan

Abstract. This paper examines strategies used in StarCraft II, a real-time strategy (RTS) game in which two opponents compete in a battlefield context. The RTS genre requires players to make effective strategic decisions. How players execute the selected strategies affects the game result. We propose a method to automatically classify strategies as rush or non-rush strategies using support vector machines (SVMs). We collected game replay data from an online StarCraft II community and focused on high-level players to design the proposed classifier by evaluating four feature functions: (i) the upper bound of variance in time series for the numbers of workers, (ii) the upper bound of the numbers of workers at a specific time, (iii) the lower bound of the start time to build a second base, and (iv) the upper bound of the start time to build a specific building. By evaluating these features, we obtained the parameters combinations required to design and construct the proposed SVM-based rush identifier. Then we implemented our findings into a StarCraft: Brood War (StarCraft I) agent to demonstrate the effectiveness of the proposed method in a real-time game environment.

Keywords: real-time strategy game · StarCraft · rush strategy · support vector machine · game log.

1 Introduction

Real-time strategy (RTS) games are popular online computer games in which two opponents compete on a battlefield. RTS game players must gather resources to develop combat strength by obtaining advanced buildings, technologies, and armies. Unlike other strategy games, such as Go and Chess, RTS game information is more complex and partially limited to the players which only can be seen by carefully observing through scouting. The complexity in RTS games covers both the number of available actions and locations to choose which contributes to the wide decisions among all possibilities [11]. Such information changes rapidly as players respond to various actions [4, 6, 12]. Therefore, players must perform multiple tasks simultaneously within a short period [17]. These characteristics contribute to an RTS game's level of difficulty [3]. In addition, such characteristics make developing artificial intelligence (AI) or bots for such games difficult [2].

RTS games can be considered a simplification of real-life environments [11]. An RTS game environment is constructed from complex [11,14,19] and dynamic

information [20] simultaneously. In an RTS game, such information changes frequently depending on the players' actions. Dealing with this type of environment is a significant challenge for AI developers. The quality of RTS game AI has been improved due to various competitions [16], such as the Student StarCraft AI Tournament,[1] the AIIDE StarCraft AI Competition,[2] and the CIG StarCraft RTS AI Competition.[3]

In RTS games, selecting effective and timely strategies is extremely important to counter an opponent's play style. Making ineffective and poorly timed decisions can lead to a strategy that hinders the player. Note that this can happen even to high-level players. With professional gamers, human players perform various strategies with good decision making and control skills based on the information they receive. How human players determine their strategy develops their own play style and results in a high win rate. In addition, it is important to learn and analyze effective human player strategies when developing RTS game AI. In consideration of these factors, we investigate the classification of StarCraft strategies as rush and non-rush strategies. A rush strategy aims to destroy the opponent early before the enemy has prepared an effective defense while a non-rush strategy is generally the opposite of rush strategy that more focus on the development (e.g., building and technology advancement). To examine player strategies, we design a support vector machine (SVM)-based model that automatically classifies strategies from StarCraft II game logs into rush and non-rush strategies. We then evaluated our findings by implementing a rush detection manager into a StarCraft: Brood War (StarCraft I) agent to examine the effectiveness of the proposed model in a real-time game environment.

2 Related Work

Many studies have investigated game prediction and analysis in StarCraft. Avontuur, Spronck, and Van Zanen [1] focused on player model prediction to distinguish the level of a player. Accordingly, Liu et al. [9] investigated a player's game style in StarCraft II using several machine learning techniques to predict player actions. Predicting player actions can help human players to determine the strategies used by other players.

Studies into the prediction of strategies have also been conducted [13, 18]. Weber and Mateas [18] used data mining techniques to create data about an opponent's constructed buildings to predict their strategy. They indicated that analyzing information about an opponent's buildings can help discriminate different strategies. Park et al. [13] used a scouting algorithm and several machine learning approaches to predict an opponent's strategy. They applied this approaches to an AI bot that recognizes an opponent's constructed building (the build order) by sending a scout. Ruíz-Granados [15] developed a model that can

[1] https://sscaitournament.com/
[2] http://www.cs.mun.ca/~dchurchill/starcraftaicomp/
[3] http://cilab.sejong.ac.kr/sc_competition/

Table 1. Game Log Data of StarCraft II (with T = Terran, Z = Zerg, and P = Protoss)

Game type	T vs. T	Z vs. Z	P vs. P	T vs. Z	T vs. P	Z vs. P	Total
Number of game logs	65	102	28	240	177	141	753

predict the winner of a StarCraft match at a specific time using replay information. In addition, Justesen and Risi [7] trained a deep neural network to learn build orders from StarCraft replays. They focused on game macromanagement to enhance a bot's competitiveness against other bots. Another study of StarCraft rush games is presented in [10] introducing features for identifying the rush games, but it does not implement any machine learning technique which examines their proposed features by using AND and OR logic feature combinations. We extend the accomplishments of these studies by exploring strategy in RTS games by focusing on rush matches. Our goal is to develop a method to collect data and identify a human player's strategies using an SVM. Furthermore, we integrated the proposed rush identification method in an agent to play real games to evaluate the correctness of the proposed rush identification method.

3 StarCraft

3.1 Overview

StarCraft is a well-known RTS game series developed by Blizzard Entertainment™. The most common match in StarCraft is the one-versus-one game [18], where the purpose is to destroy another player's units. Its game expansion, StarCraft: Brood War, turns into the version that is played competitively and becomes a subject of AI development through bot competition [8]. StarCraft series was followed by the release of StarCraft II: Wings of Liberty with two expansions, Heart of the Swarm and Legacy of the Void. Both in StarCraft and StarCraft II, there are three races that can be chosen in the game environment, i.e., the Terran, Zerg, and Protoss, and each race has unique but comparable strengths and weaknesses [11]. In these games, players must collect resources, build structures, and train armies to compete in battle. Moreover, to perform competitively, the games demand good decision making and control skills.

3.2 Rush Strategy

Rush strategies are used to initiate a quick attack against the opponent. The goal of a rush strategy is to destroy the opponent in the early stages of the games, i.e., before the opponent has prepared an effective defense. Note that players who engage rush strategies often sacrifice the ability to improve their base and upgrade to advance technology because they quickly spend many resources preparing an army and constructing buildings. Rush strategies can be used in any type of RTS game to defeat an opponent as quickly as possible.

Table 2. Dataset for Evaluation

Logs	Rush strategy	Non-rush strategy	Total
Number of game logs	137	616	753

4 Resources and Datasets

We collected game replays of one-versus-one StarCraft II games from a website, namely *spawningtool.com*[4]. To standardize our examination, we only used game replays from *StarCraft II: Legacy of the Void* because other game versions have different building and army characteristics. All replay files were extracted as human-readable log files using a library that obtains StarCraft II replay information, i.e., *SC2Reader*[5]. A game log contains the list of time and actions performed by the players in the game such as training a unit, and creating a building shown as following sample [*04.13*][*PlayerName*][*Unit born UnitName*].

To examine the rush strategy, we received an assistance from a StarCraft player in Diamond league to manually classified each game as a rush or non-rush game. Note that we focused on only high-level league games i.e., Diamond, Master, and Grandmaster leagues, because it was necessary to collect data for successful strategies from high-level players. From the collected total 5,150 data, we obtained 753 game logs under this condition which the distribution of all race matches of this data is shown in Table 1. Each sample consists of a single player's game log that could be categorized as a game with a rush or non-rush strategy (Table 2).

5 Time Series Changes in Number of Workers

We propose several features closely related to the number of workers of each player. These features are based on the observation of rush games. Rush strategy players do not consume a lot of resources on infrastructure, such as workers, building upgrades, technologies, and resource extractors. Figure 1 compares typical average time series changes in the numbers of workers between the rush and non-rush strategies. As can be seen, there is a significant difference in the number of workers in these different strategies. Here, rush strategy players train a moderate number of workers and do not train additional workers in the next phase of the game; thus, there is no change in time series of the number of workers. In contrast, non-rush strategy players continue producing a significantly greater number of workers compared to rush strategy players. By considering typical situations in rush and non-rush strategies, we designed features based on the variance of the time series of the number of workers and the number of workers at a specific time. Note that we consider only the number of workers a player has trained up to a given time rather than the number of workers a player has

[4] `http://www.spawningtool.com`
[5] `https://github.com/GraylinKim/sc2reader`

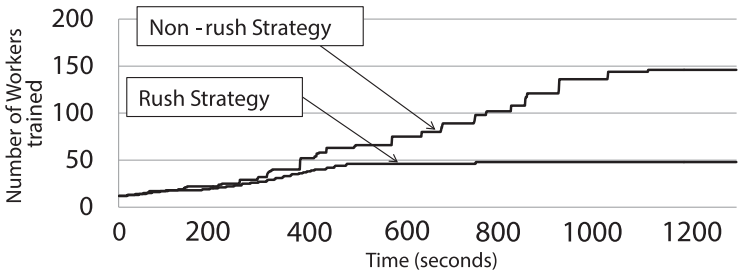

Fig. 1. Time Series Changes in the Numbers of Workers of Rush/ Non-rush Strategies

at a certain time. This means the number of workers a player has at a certain time is always affected by how many workers survive in the battle. Thus, we only consider the number of workers the player has direct control over.

6 Game Log Features

Here, let g be a game replay comprising the game logs g_1 and g_2 of players 1 and 2, respectively, and let x be either game log g_1 or g_2:

$$g = \langle g_1, g_2 \rangle,\ x = g_1 \text{ or } g_2$$

Table 3 defines the four types of features, i.e., the upper bound of variance of the time series of the number of workers (vw), the upper bound of the number of workers at a specific time (nw), the lower bound of the start time of building a second base (b), and the upper bound of the start time to build a specific building (sp).

6.1 Upper Bound of Variance of Time Series of Number of Workers

The feature function $f_{vw}(x; u_0, d_0, e_0)$ of game log x examines whether the variance of the time series of the number of workers (vw) for time duration d_0 at end time e_0 of the variance calculation satisfies the upper bound u_0 as follows.

$$f_{vw}(x; u_0, d_0, e_0) = (x.f^v_{vw} \leq u_0) \wedge (x.f^d_{vw} = d_0) \wedge (x.f^e_{vw} = e_0)$$

The variance of the time series in the numbers of workers is measured at a time interval of one minute.

6.2 Upper Bound of Number of Workers at a Specific Time

The feature function $f_{nw}(x; n_0, t_0)$ of game log x examines whether the number of workers (nw) at time t_0 satisfies the upper bound n_0 as follows.

$$f_{nw}(x; t_0, n_0) = (x.f^t_{nw} = t_0) \wedge (x.f^n_{nw} \leq n_0)$$

6.3 Lower Bound of Start Time to Build Second Base

In addition to the feature functions discussed in the previous sections, which are related to the number of workers of each player, we also propose a third feature that is related to the start time of building a second base. The existence of a base (a building for collecting resources) in the field is important in the game. To collect more resources, the players should expand their base by building a second base as quickly as possible at a location that potentially provides more resources. In the case of non-rush strategies, resources are more important compared to rush strategies because such resources are required to build a second base immediately and safely. Players who use the rush strategy do not necessarily build a second base as early as possible. Accordingly, it is expected that a rush strategy player will build a second base over a certain time. Based on this observation, this section introduces the lower bound of the start time of building a second base. The feature function $f_b(x; t_0)$ of the lower bound of the second base build start time (b) of game log x examines whether the start time satisfies the lower bound t_0 as follows.

$$f_b(x; t_0) = (x.f_b^t \geq t_0)$$

6.4 Upper Bound of Start Time to Build a Specific Building

The fourth feature is also related to building information. Each race in a rush game must prioritize specific buildings as early as possible to enable a rush attack. Those buildings differ based on the build start time for each race. In the case of Zerg, the starting time to build first *Spawning Pool* is used, and the starting time to build second *Barracks* and second *Gateway* for Terran and Protoss respectively are used to examine this feature function. We observed the timing of these specific buildings (sp), and each player tended to build them before reaching a particular time. In the rush attack, the timing to build these buildings for each race is observed in the beginning of the game. The feature function $f_{sp}(x; t_0)$ of the upper bound of the start time to build the specific building (sp) satisfies the upper bound t_0 as follows.

$$f_{sp}(x; t_0) = (x.f_{sp}^t \leq t_0)$$

7 Overall Design

Parameter combinations of the feature functions f_{vw}, f_{nw}, f_b, and f_{sp} were examined to determine the parameter combinations which possess the maximum recall, precision and f-measure. For f_{vw}, combinations of parameters were examined by changing v_0 from 0 to 2, d_0 from 60 to 300, and e_0 from 240 to 360. For f_{nw}, combinations of parameters were examined by changing t_0 from 300 to 600 and n_0 from 25 to 40. For f_b, the parameter was examined by changing t_0 from 60 to 360. Finally, for f_{sp}, the parameter was examined by changing t_0 from 20

Table 3. Features of Game Log x

Features	Variables of x	
Upper bound of variance of time series of numbers of workers $f_{vw}(x; u_0, d_0, e_0)$ $= (x.f^v_{vw} \leq u_0) \wedge (x.f^d_{vw} = d_0) \wedge (x.f^e_{vw} = e_0)$	$x.f^v_{vw}$	Variance of x
	$x.f^d_{vw}$	Time duration of calculating variance [s]
	$x.f^e_{vw}$	End time of calculating [s]
Upper bound of number of workers at a specific time $f_{nw}(x; t_0, n_0) = (x.f^t_{nw} = t_0) \wedge (x.f^n_{nw} \leq n_0)$	$x.f^t_{nw}$	Specific time [s]
	$x.f^n_{nw}$	Number of workers
Lower bound of start time of building a second base $f_b(x; t_0) = (x.f^t_b \geq t_0)$	$x.f^t_b$	Start time of building the second base [s]
Upper bound of start time to build a specific building $f_{sp}(x; t_0) = (x.f^t_{sp} \leq t_0)$	$x.f^t_{sp}$	Start time of building a specific building [s]

to 360. The number we selected for each parameter of the feature functions is based on our observation to the logs of the rush games.

We first divided our dataset into 10 subsets of equal size to perform 10-fold cross validation. Each subset was used sequentially as test data, where the remaining 90% was used as training data. From the training data of each fold, the parameter combinations of each feature functions f_{vw}, f_{nw}, f_b and f_{sp} were identified from the combinations that yielded the maximum recall, precision, and f-measure values as shown in Figure 2. Moreover, Table 4 shows the parameter combinations of each fold with maximum recall, precision, and f-measure. Note that variables depicted in the tuples in Table 4 follow the order of Table 3. Using this procedure, each feature function generated three parameters combinations, resulting in a total of 12 parameter combinations for each fold. Formally, those 12 parameters are as follows.

$$F = \{f^r_{vw}, f^p_{vw}, f^f_{vw}, f^r_{nw}, f^p_{nw}, f^f_{nw}, f^r_b, f^p_b, f^f_b, f^r_{sp}, f^p_{sp}, f^f_{sp}\}$$

Here, F is a set of parameters combinations generally constructed of a set of feature functions f_{vw}, f_{nw}, f_b, and f_{sp}, where r, p, and f are maximum recall, precision, and f-measure respectively. We created and used a feature vector constructed of these 12 features. Eventually, our design had 10 different sets of parameter combinations, which were used to train the SVM classifier.

Table 4. Parameter combinations with maximum recall, precision, and f-measure

Fold	f^r_{vw}, f^p_{vw}, f^f_{vw}	f^r_{nw}, f^p_{nw}, f^f_{nw}	f^r_b, f^p_b, f^r_b	f^r_{sp}, f^p_{sp}, f^f_{sp}
1	(2, 120, 300), (0, 220, 240), (2, 195, 300)	(300, 40), (520, 25), (390, 31)	(60),(293),(118)	(96), (24), (37)
2	(2, 120, 300), (0, 220, 240), (2, 165, 300)	(300, 40), (520, 25), (390, 31)	(60),(293),(139)	(96), (24), (37)
3	(2, 120, 340), (0, 290, 340), (2, 150, 260)	(310, 40), (520, 25), (390, 31)	(60),(293),(118)	(90), (24), (37)
4	(2, 120, 340), (0, 290, 340), (0.5, 150, 300)	(300, 40), (530, 25), (390, 31)	(60),(293),(118)	(96), (24), (46)
5	(2, 120, 340), (0, 285, 340), (1, 135, 320)	(300, 40), (520, 25), (390, 31)	(60),(291),(118)	(96), (24), (37)
6	(2, 120, 340), (0, 290, 340), (1, 135, 320)	(300, 40), (410, 25), (390, 31)	(62),(293),(139)	(96), (25), (37)
7	(2, 120, 320), (0, 290, 240), (2, 165, 300)	(300, 40), (520, 25), (420, 32)	(60),(293),(118)	(96), (24), (37)
8	(2, 120, 340), (0, 240, 260), (2, 165, 300)	(300, 40), (410, 25), (390, 31)	(70),(293),(119)	(96), (24), (37)
9	(2, 120, 360), (0, 285, 340), (2, 155, 300)	(300, 40), (520, 25), (390, 31)	(60),(293),(118)	(96), (31), (37)
10	(2, 120, 300), (0, 285, 340), (2, 155, 300)	(300, 40), (520, 28), (390, 31)	(60),(293),(118)	(96), (24), (37)

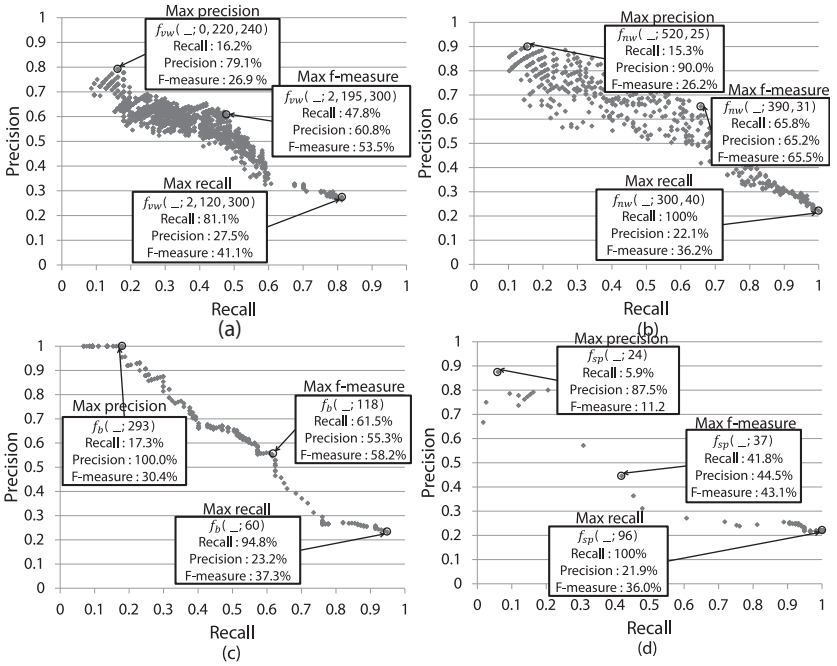

Fig. 2. Parameters combinations of the first fold of 10-fold cross-validation of feature functions (a)f_{vw}, (b)f_{nw}, (c)f_b, and (d)f_{sp}

8 Evaluation

8.1 Experimental Setup

We attempted to design a model for rush game classification from the game logs using an SVM. We applied an SVM technique to identify whether a game log includes a rush strategy, and we used an SVM library provided by LIBSVM[6]. The number of instances in this experiment is shown in Table 2. Using only the training data in each iteration, we found the parameter combinations with maximum recall, precision and f-measure, which we applied as SVM features to both the training and test sets. These procedures were replicated 10 times.

8.2 Results

We used confidence to calculate the performance of each fold of the proposed approach using recall and precision. We then plotted the average performance curve based on this calculation (Figure 3). The curve in Figure 3 shows 11 plot points (from 0 to 100) that represent the average performance of all folds. We

[6] https://www.csie.ntu.edu.tw/ cjlin/libsvm/

generalized the recall value of each fold to the closest position among these 11 points. Figure 3 also shows the recall-precision curves of the proposed design compared to four alternatives. Each alternative curve was produced by removing each set of parameter combinations of feature functions f_{vw}, f_{nw}, f_b, and f_{sp} from the evaluation. By comparing the proposed design to its alternatives, it was found that the proposed design was outperformed slightly by an alternative design constructed without feature functions f_{vw}. This result indicates that a design using parameter combinations of $f^r_{nw}, f^p_{nw}, f^f_{nw}, f^r_b, f^p_b, f^f_b, f^r_{sp}, f^p_{sp}, f^f_{sp}$ demonstrates better performance than the proposed design with all four feature functions. We selected this alternative's feature as the optimal feature function (Figure 3).

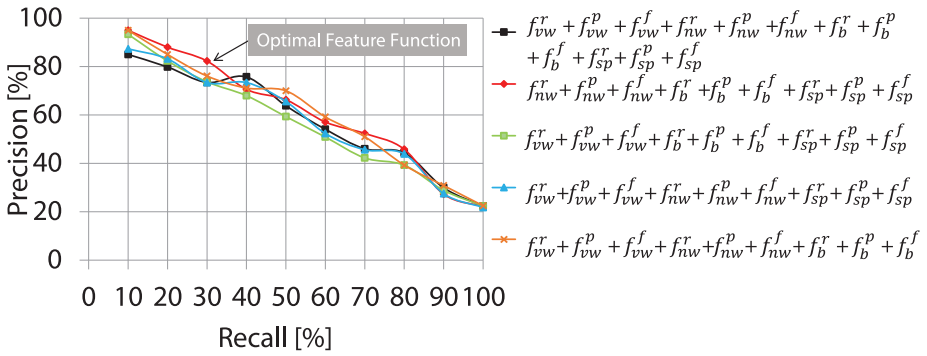

Fig. 3. Recall and precision curve of the combinations of f_{vw}, f_{nw}, f_b, and f_{sp}.

Based on these results, we further examined the optimal feature function of $f^r_{nw}, f^p_{nw}, f^f_{nw}, f^r_b, f^p_b, f^f_b, f^r_{sp}, f^p_{sp}, f^f_{sp}$ by comparing this design by removing each remaining parameter combinations f_{nw}, f_b and f_{sp} from the evaluation. The results are shown in Table 4. The overall recall-precision performance shows that the proposed design using combinations of $f^r_{nw}, f^p_{nw}, f^f_{nw}, f^r_b, f^p_b, f^f_b, f^r_{sp}, f^p_{sp}, f^f_{sp}$ demonstrates the highest recall and precision among all alternatives. We received worse performance when evaluating alternative design without including feature function f_{nw}. This may be because the difference in the number of workers at a specific time in a rush game provides meaningful information to classify the rush game. Moreover, the build time and feature function f_{nw} information contributes to the performance of the proposed method. The results (Figure 4) indicates that there was a significant correlation to the parameter combinations with maximum recall, precision and f-measure of the features functions f_{nw}, f_b, and f_{sp}. The result indicate that the proposed design with these three parameter combinations worked better than using all four feature functions. Therefore, the proposed design could possibly be effective at identifying rush games in collections of RTS game logs.

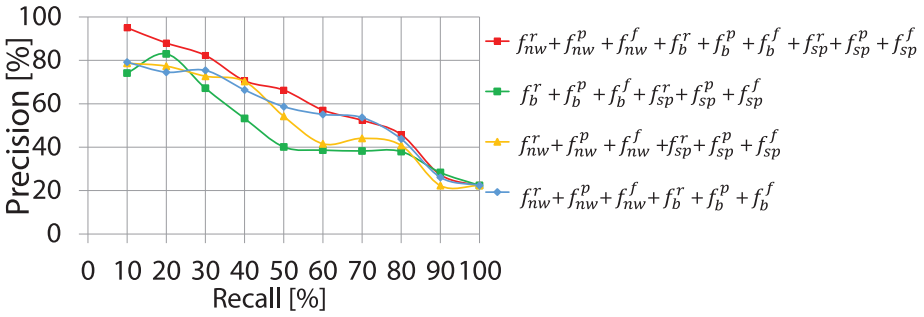

Fig. 4. Recall and precision curve of the combinations of f_{nw}, f_b, and f_{sp}.

8.3 Incorporating Rush Identifier to StarCraft: Brood War Agent

We further evaluated the proposed method by developing an agent called *RushIdentifierBot* in StarCraft: Brood War based on the UAlbertaBot[7] architecture as the seed bot. The UAlbertaBot was implemented using a build order planning system that focuses on optimizing build order problems in StarCraft [3] and unit combat scenarios that result in unit actions as the outcome [4, 5]. There also exists a StarCraft II bot which is based on the architecture of UAlbertaBot, CommandCenter[8]. We could possibly do the same things on StarCraft II since the source of our bot architecture is the same. Note that we selected using StarCraft: Brood War rather than StarCraft II due to the availability of opponent bots. We selected an existing bot framework because it has been integrated using the basic functions required to run the game. Thus, we could focus on our purpose, i.e., improving rush strategy identification in RTS games. The RushIdentifierBot has a strategy changing ability integrated into a module call the Rush Detection Manager. This module detects the opponent's rush action in the early of the game state using scouting information to identify rush or non-rush strategies.

The feature functions f_{nw}, f_b, and f_{sp} of the proposed model were integrated into the rush detection manager. Here, we used these three feature functions and removed feature function f_{vw} from the development of the rush detection manager because feature function f_{vw} reduced performance. Moreover, due to computational complexity, we did not implement the SVM model in our agent; however, we did implement a rule-based system for the rush detection manager using the optimum result obtained by the SVM. We tuned the parameters combinations of the three features f_{nw}, f_b, and f_{sp} by trial and error using held-out data, the additional data used for trial error, to determine the number of wins and loses of our bot. The data were based on the information obtained in a real-time game played by our agent against several bots that participated in the

[7] `https://github.com/davechurchill/ualbertabot`
[8] `https://github.com/davechurchill/commandcenter`

Table 5. Judgment Correctness

Bot's Judgment		Correct Judgment		Incorrect Judgment		Total	
Win or Loss		Win	Loss	Win	Loss	Win	Loss
Bot's Judgment	Rush	19	12	0	0	19	12
	Non-rush	31	26	0	3	31	29
	No judgment	0	0	0	9	0	9
Total		50	38	0	12	50	50

2017 AIIDE StarCraft AI Competition. Note that the trial and error game data are not included in the evaluation results.

Furthermore, we evaluated the correctness of our bot's rush identification by playing 400 games against 15 well-known bots from the 2017 AIIDE StarCraft AI Competition: *Arrakhammer, cpac, IceBot, Iron, Juno, KillAll, LetaBot, McRave, MegaBot, Microwave, Overkill, Sling, Steamhammer, AIUR,* and *Tyr.* Note that these bots use different races. We randomly selected 50 games for each win and loss sample from the 400 games played by our bot in this evaluation. Table 5 shows the correctness of the rush identification function in our agent. The judgment of rush or non-rush by our bot was made during the game in the range from minute 2 until 6. The majority of the sample in Table 5 shows that our bot could, for the most part, judge opponent rush actions correctly. However, there were only three samples our bot judged incorrectly. All winning cases were observed when our bot could make correct judgments in the real-time game. Note that, even though our bot could judge rush or non-rush actions correctly, there were still games in which our bot lost. This could have occurred because our bot did not appropriately adapt to the opponent's late-game strategy. In addition, when our bot could not make a judgment, we categorized this situation as an incorrect judgment. Such cases occurred because our bot did not have any information about the opponent's state such as failure to find the location of the opponent's base.

9 Conclusion

This study has proposed a method to identify rush strategies in an RTS game using replay log data. We collected game replays from a StarCraft II community website to identify rush and non-rush strategies. Note that we primarily focused on rush games played by high-level StarCraft II players. We examined 12 parameter combinations of our four feature functions, i.e., f_{vw}, f_{nw}, f_b, and f_{sp}. We found that using f_{nw}, f_b, and f_{sp} features showed better performance than using all four features. Therefore, we used these features to design the SVM used to identify rush strategies. Further evaluation were performed by implementing our rush strategy identification in a StarCraft I agent. We evaluated the correctness of our bot identification function in games against 15 well-known bots.

Even though our bot could identify rush or non-rush actions, it could not defeat all of the opponent bots, which may have been due to lack of appropriate late-game strategy decisions. Note that rush strategies are only employed in the early stage of a game, and in longer games, do not provide any advantages to the overall strength of bots. Thus, it would be beneficial to further evaluate late-game strategies to improve bots performance.

References

1. Avontuur, T., et al.: Player skill modeling in StarCraft II. In: Proc. the 9th AIIDE. pp. 2–8 (2013)
2. Buro, M., Furtak, T.M.: Rts games and real-time AI research. In: Proc. BRIMS. vol. 6370 (2004)
3. Churchill, D., Buro, M.: Build order optimization in StarCraft. In: AIIDE. pp. 14–19 (2011)
4. Churchill, D., Buro, M.: Incorporating search algorithms into rts game agents. In: AIIDE (2012)
5. Churchill, D., et al.: Fast heuristic search for rts game combat scenarios. In: AIIDE. pp. 112–117 (2012)
6. Gaudl, S.E., et al.: Behaviour oriented design for real-time-strategy games. In: FDG. pp. 198–205 (2013)
7. Justesen, N., S.Risi: Learning macromanagement in Starcraft from replays using deep learning. In: CIG. pp. 162–169 (2017)
8. Lewis, J., et al.: A corpus analysis of strategy video game play in StarCraft: Brood War. In: CogSci. vol. 33 (2011)
9. Liu, S., et al.: Player identification from RTS game replays. In: Proc. the 28th CATA. pp. 313–317 (2013)
10. Oh, H., et al.: Identifying the rush strategies in the game logs of the real-time strategy game StarCraft-II. In: Proc. 31st Annual Conf. JSAI (2017)
11. Ontanón, S., et al.: A survey of real-time strategy game AI research and competition in StarCraft. IEEE T-CIAIG **5**(4), 293–311 (2013)
12. Ontanón, S., et al.: Rts AI problems and techniques. Encyclopedia of Computer Graphics and Games pp. 1–12 (2015)
13. Park, H., et al.: Prediction of early stage opponents strategy for StarCraft AI using scouting and machine learning. In: Proc. WASA. pp. 7–12 (2012)
14. Robertson, G., Watson, I.: A review of real-time strategy game AI. AI Magazine **35**(4), 75–104 (2014)
15. Ruíz-Granados, A.S.: Predicting the winner in two player StarCraft games. In: Proc. the 2nd CoSECiVi. pp. 24–35 (2015)
16. Čertický, M., Churchill, D.: The current state of StarCraft AI competitions and bots. AIIDE (2017)
17. Usunier, N., et al.: Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. arXiv preprint arXiv:1609.02993 (2016)
18. Weber, B.G., Mateas, M.: A data mining approach to strategy prediction. In: Proc. the 5th CIG. pp. 140–147 (2009)
19. Weber, B.G., et al.: Building human-level AI for real-time strategy games. In: AAAI Fall Symposium: Advances in Cognitive Systems. vol. 11, p. 01 (2011)
20. Wintermute, S., et al.: Sorts: A human-level approach to real-time strategy AI. Ann Arbor **1001**(48), 109–2121 (2007)