In-Game Action List Segmentation and Labeling in Real-Time Strategy Games

Wei Gong, Ee-Peng Lim, Palakorn Achananuparp, Feida Zhu, David Lo and Freddy Chong Tat Chua

***Abstract*—In-game actions of real-time strategy (RTS) games are extremely useful in determining the players' strategies, analyzing their behaviors and recommending ways to improve their play skills. Unfortunately, unstructured sequences of in-game actions are hardly informative enough for these analyses. The inconsistency we observed in human annotation of in-game data makes the analytical task even more challenging. In this paper, we propose an integrated system for in-game action segmentation and semantic label assignment based on a Conditional Random Fields (CRFs) model with essential features extracted from the in-game actions. Our experiments demonstrate that the accuracy of our solution can be as high as 98.9%.**

I. Introduction

Real-time strategy (RTS) game is a popular online game genre in which players gather necessary resources to train army units and build bases to fight against opponents. To maximize the chance of winning in RTS games, one needs to learn the right strategies. Hence, it is not a surprise many players turn their attention to game commentaries that explain the dos and don'ts of game playing using previously recorded games for illustration. For example, Sean 'Day9' Plott, a famous commentator of the highly popular RTS game *StarCraft II*[1], has attracted over 45 millions views on his YouTube Channel[2].

From the RTS game developer perspective, to attract and retain players, it is equally important to help their players understand the games and learn the right strategies to play well. Today's RTS games, however, have only limited capacity to offer such assistance to their players other than a good help menu and user guide. The current best way for players to learn the games is still by trial and error, a potentially tedious and time-consuming process.

Fortunately, the recent advances in database systems and collaborative web sites, have made it possible to record massive amount of data generated by real RTS games (often known as *game replays*), and share them at game data repository websites. One can then apply data analytics to discover game strategies and to evaluate their effectiveness in improving gaming performance and experience. The knowledge gained can be further used to guide individual players to play better and help game developers design better games.

Wei Gong, Ee-Peng Lim, Palakorn Achananuparp, Feida Zhu, David Lo and Freddy Chong Tat Chua are with the School of Information System at Singapore Management University. 80 Stamford Road, Singapore (email: `wei.gong.2011@smu.edu.sg`, `eplim@smu.edu.sg`, `palakorna@smu.edu.sg`, `fdzhu@smu.edu.sg`, `davidlo@smu.edu.sg`, `freddy.chua.2009@smu.edu.sg`)

[1]http://sea.blizzard.com/games/sc2/

[2]http://www.youtube.com/user/day9tv/

In this paper, we analyze the in-game actions of a popular RTS game, *StarCraft II: Wings of Liberty*, developed and released by Blizzard Entertainment. StarCraft II (or SC2) is selected because (1) the popularity of SC2 makes it easy to find highly active players for the purpose of human annotation; (2) SC2 replay files are publicly available for downloading from several websites; (3) the highly complicated gameplay in SC2 has given rise to a wealth of sophisticated strategies from its players; and (4) comparing to the original *StarCraft*[3], SC2 is a newer game with a greater variety of gameplay mechanics. Hence, the game is not yet fully understood by the players and the gameplay strategies are still evolving, which makes SC2 more interesting and challenging for analysis. Even as this paper focused on SC2 only, the same problem and proposed techniques will be applicable to other popular RTS games such as *League of Legends*[4], and *DOTA2*[5].

A replay file in SC2 records in temporal order all actions performed by the players, including mouse movements and keyboard inputs. These mouse and keyboard actions are recorded as atomic actions, such as training a unit, selecting a building or attacking some facility. The sequence of timestamped actions performed by one player in a game is called an *action list*.

While unstructured action lists are hardly adequate to reveal the player's intention and strategy, it is much easier when consecutive atomic actions are grouped into logical segments assigned with meaningful semantic labels. Our objective is thus to partition each SC2 action list into segments and give each segment a label which represents the collective meaning of the actions in this segment. Knowing these action segments and their semantic labels will give us an idea how the player executes her strategy in the game.

Formally, given an action list $L = (a_1, a_2, ..., a_m)$ with m actions performed by one player, a segmentation S of L is defined as k non-overlapping segments, denoted as $S = \{s_1, s_2, ..., s_k\}$, where for $1 \leq i \leq k$, $s_i = (a_{b_i}, a_{b_i+1} ..., a_{b_i+l_i})$, $1 \leq b_i < b_{i+1} \leq m$ and $\sum_{i=1}^{k} l_i = m - k$. Let $A = \{\alpha_1, \alpha_2, ..., \alpha_h\}$ be a set of h unique labels. Our problem is to find all the segments $s_1, s_2, ..., s_k$, and assign a label $\alpha_{s_i} \in A$ for each segment s_i, $1 \leq i \leq k$. In this problem formulation, we assume that the label set A is given. To obtain this label set, we actually need some expert knowledge about the SC2 game which is covered in Section

[3]http://sea.blizzard.com/games/sc/

[4]http://lol.garena.com/playnow.php/

[5]http://www.dota2.com/

III-B.

The task of segmentation and labeling sequential data has been studied in many fields, including bioinformatics [6], natural language processing [2] and speech recognition [14]. The segmentation techniques include Hidden Markov Models (HMMs) [15], Maximum Entropy Markov Models (MEMMs) [11], and Conditional Random Fields (CRFs) [9]. Since CRFs are known to outperform both HMMs and MEMMs on a number of real-world sequence segmentation and labeling tasks [9], [13], we use CRFs to automatically segment and label game action lists in our framework.

Several challenges are unique in this in-game action list segmentation and labeling task. Firstly, to the best of our knowledge, there are no publicly available labeled action lists to be used for training. The manual process of labeling takes much time and effort and the labeling agreement between annotators in our experiment has been shown to be far below what we considered to be useable. Secondly, the noise level in the raw in-game action lists is very high. This prevents accurate segmentation and labeling. Our experiments show that 80% of actions in the actions lists can be considered as spams. The spam actions are generated in various ways, such as accidental keypress, repeated use of trivial game commands to inflate personal statistics, i.e. number of actions performed per minute, etc. Finally, we need to identify the features to be used in segmentation and labeling. None of the above tasks has been studied for the segmentation task in the past.

In our literature survey, we found several prior works on RTS games focusing on discovering the build order based strategies [20], [21] and incorporating them into the artificial intelligence (AI) game agents [12], [22]. However, the above works make assumption about the way the games will be played and only focus on a small subset of in-game actions (build order). There is relatively little work on mining game strategies from the entire set of in-game actions, and using the mined strategies to explain the game matches. In [5], action list segmentation and labeling was also studied but the work focused on fixed length segmentation approach over the in-game actions related to build order. The work assumes equal length segments of 30 seconds each, and applies HMMs to learn the segment labels. Unlike our work below, this segmentation method is not data-driven and does not consider human annotated ground truths in training and evaluation.

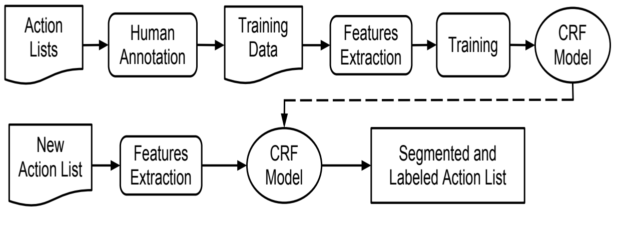


Fig. 1. Framework for Our Approach

A. Contributions

The in-game action segmentation and labeling task is novel. By identifying and addressing the associated challenges, we contribute to game analytics research as follows:

1) We proposed a framework (as shown in Figure 1) to solve the action list segmentation and labeling problem. This framework has two phases, namely model training and model application. In model training, we collect raw action lists, recruit annotators to segment and label action lists so as to obtain the training data, extract representative features from the training data, and train a CRF model. In model application, features are extracted from a new action list, which is then segmented and labeled by the trained CRF model.
2) We have developed a prototype annotation tool known as **Labelr**[6] to collect segmentation and labeling data from SC2 players so as to derive the ground truth data.
3) We have devised a simple heuristics to filter spurious actions from the action lists. We show that our trained CRF model achieves high accuracy in the segmentation and labeling task.

B. Outline of Paper

The outline of this paper is as follows. Section II describes the in-game data of SC2. Section III introduces the dataset we collected. Section IV describes our proposed CRF based segmentation and labeling method. Section V presents our experimental results. Section VI concludes by summarizing our work and presenting the future work.

II. Overview of StarCraft II

A. StarCraft II Mechanics

SC2 is a military science fiction RTS game developed by Blizzard Entertainment. In SC2, players observe the game actions from a topdown perspective. They are required to collect resources, develop technologies, build up their army and use them to destroy the opposing player's. Players can choose to play as one of the three unique races, *Terran*, *Protoss*, and *Zerg* when game starts. Each race is designed to be played differently with its own sets of units and buildings.

When the game starts, players issue commands to the units and buildings under their control in real-time through mouse clicks and keyboard shortcuts. This is different from turn-based games, such as chess, where players take turn to move their pieces.

Figure 2 shows the types of objects in SC2. There are two kinds of resources: *minerals* and *gas*. Minerals are the basic form of currency required in every training, building, researching and upgrading actions. Gas is the rarer form of currency used in the training and construction of more advanced units and buildings as well as upgrading technology. There are three types of units: *worker*, *combat*, and *production unit*. Workers are responsible for collecting resources and constructing buildings. Although they can also attack other

[6]http://202.161.45.174/sc2annotation

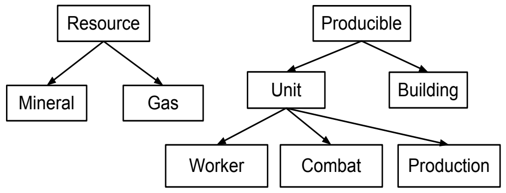


Fig. 2. Objects in SC2

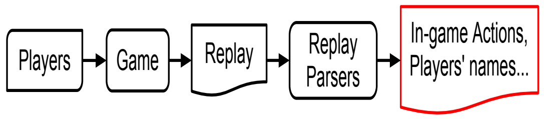


Fig. 3. Collecting replays and action lists

TABLE I
A PARTIAL ACTION LIST PERFORMED BY ONE PLAYER FROM A SC2 REPLAY

Time (mins)	Player Action
0:00	Select Command Center (10288)
0:00	Train SCV
0:00	Train SCV
0:00	Train SCV
0:01	Select SCV x6 (1028c, 10290, 10294, 10298, 1029c, 102a0), Deselect all
0:02	Right click; target: Mineral Field (100b4)
...	...
0:07	Select Command Center (10288), Deselect all
0:08	Right Click; target: Mineral Field (10070)

units, they are much weaker than the regular combat units. Combat units are comprised the main army of the players. Each combat unit has its own strengths and weaknesses against different types of units. There exists a special type of units, called production units, such as Zerg larva, which are static and only used to produce other units. Buildings are mainly used as the production facilities to train different units. Some building can also be used to upgrade certain technology, improving the efficiency of the economy and the effectiveness of combat units.

The mechanics in SC2 requires players to perform both micro-management (micro) and macro-management (macro) actions. The micro actions are those involving tactical movement, positioning, and maneuvering of individual or groups of units to maximize their damage on opponents and minimize the damage received. On the other hand, the macro actions involve improving and maintaining the overall economic and military profile of the players. These include a constant production of worker and combat units, upgrading the technology levels, etc. Highly skilled players are typically those who multitask effectively between micro-management and macro-management.

Competitive online multiplayer mode is what makes SC2 a highly popular game. In this mode, players can find other human players who are near their skill level to compete against using Battle.net, an official online multiplayer platform developed by Blizzard. At the end of each multiplayer game, players' performance will be evaluated automatically by Battle.net and skill ratings will be awarded to the players. Similar to chess, players' skills are categorized into different leagues. The leagues ranked from the lowest to the highest include Bronze, Silver, Gold, Platinum, Diamond, Master, and Grandmaster. Battle.net's matchmaking system will try to pair players of comparable skill levels to play against each other.

In this paper, we define a game match to be an instance of SC2 game played by at least two human players, such that this match ends with a win or lose outcome for each player. As shown in Figure 3, when a game is over, all game match data can be saved as a replay. The replay file records everything that SC2 in-game renderer needs to replay the corresponding game. Therefore, the replay file logs rich information about the game, such as chat, map name, players' name, and all actions performed by each player from the time the game starts till it ends. After collecting the replay files, we can use the replay parsers such as *phpsc2replay* [19] and *sc2gears* [1] to extract all the action data in them.

B. Action List Representation

Given a replay R, an action list $L = (a_1, a_2, ..., a_m)$ is the sequence of actions performed by a player p in R in temporal order. m is the number of actions in L. Each action a_i in L has a corresponding timestamp t_i, where $t_1 \leq t_2 \leq ... \leq t_m$.

A partial example action list is shown in Table I. Each line in this table represents an action which includes timestamp and the corresponding action. Each action consists of action type and target. For example, from the first action in Table I, 'Select Command Center (10288)', we can identify the action type as 'Select', and the target as 'Command Center' (which is a type of buildings in the Terran race). In this action, the alphanumeric string (10288) is the ID for previous object (Command Center). In SC2, every object has a unique ID. In addition, certain actions are automatically generated by a game client. For example, we can see 'Deselect all' at the end of 0:01 (minutes) and 0:07 select actions, which means that if a player is selecting something at a certain time, the game client will automatically deselect all the objects that the player selected before.

After selecting the command center, the player trains three SCVs (which are the workers of the Terran race) at time 0:00, selects six SCVs at time 0:01, and right clicks a mineral field at time 0:02. After some other actions, the player then selects the command center at time 0:07, and right clicks a mineral field at time 0:08.

C. Action List Segments

Consider the action list in Table I. Although we understand the meaning of each single action, we still do not know what does this player really want to do, such as what is the difference between the actions at time 0:02 and time 0:08 (both are right clicking a mineral field)? Why does the player control SCVs? What is the player's purpose of selecting a command center? If we group actions at time 0:01 and 0:02, and group actions at time 0:07 and 0:08, at this point in the game, the player tried to develop her economy by training more workers and ordering them to harvest minerals. As new workers are produced from the command center, they automatically go to a specific mineral field set by the player's rally point. Thus, to make sense of an individual, we must also consider the context in which the action takes place. This is similar to natural language understanding, where the meaning of a word can be inferred from other neighboring words.

In the example, we may group actions at time 0:01 and 0:02 into a segment, and assign a label 'mine'. For the segment containing actions at time 0:07 and 0:08, the appropriate label is 'set rally point'.

III. Dataset and Segment Labeling

A. Data Collection

Several websites are dedicated to host SC2 replays. These sites are used by SC2 players to share selected gameplays with the public. We developed a web crawler to collect replays from *GameReplays.org*[7], a SC2 replay repository site. A total of 866 replays of SC2 games played by one player against another player (1 vs. 1) were downloaded, from which we obtained 1732 action lists.

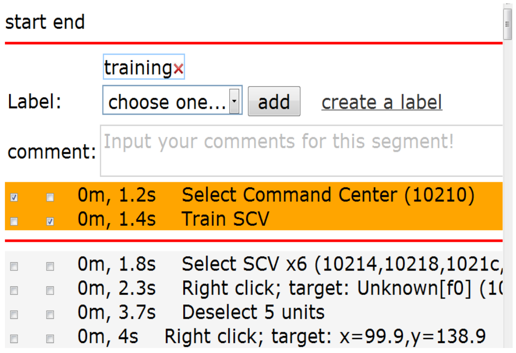
B. Manual Segmentation and Labeling

1) Human Annotation Tool: A 20-minute long SC2 game normally contains thousands of in-game actions. This makes labeling the action lists quite cumbersome for human annotators. To facilitate the process, we have developed a web-based human annotation tool called **Labelr**.

As shown in Figure 4, Labelr provides necessary web user-interface functions for an annotator to organize, segment, and label action lists with minimum effort. For example, the annotator can simply assign the start and end points to split the action sequence. Then, she can select from a default list (training, building, mining, etc.) or create an appropriate label for the corresponding segment. To aid the annotator in making sense of some complex in-game sequence, Labelr also provides a link to download the actual binary replay file which can be played back in the game client. All annotation data from each annotator are stored in a relational database.

2) General Players Annotation: Two business-major undergraduate students who have extensive knowledge in SC2's gameplay mechanics and have been competing in high level online leagues (Diamond & Platinum) for over a year were hired to label the data using Labelr. They each labeled five common action lists. From their feedback, we learned that segmentation and labeling on action list is time consuming. On average, it took an annotator three hours to label a 20-minute game. This is because the SC2 action list is very long: the average game length of the five labeled action lists is 23.4 minutes involving an average of 4628 actions. To segment and label one action list, annotators have to go through all the

[7]www.gamereplays.org/starcraft2/replays.php?game=33



0m, 1.2s Select Command Center (10210)
0m, 1.4s Train SCV

0m, 1.8s Select SCV x6 (10214,10218,1021c,
0m, 2.3s Right click; target: Unknown[f0] (1(
0m, 3.7s Deselect 5 units
0m, 4s Right click; target: x=99.9,y=138.9

Fig. 4. Part of screenshot of Labelr

TABLE II
Raters agreement, presented by Fleiss' Kappa

(a) Kappa of replay annotation

Action list id	Kappa value
1	0.43
2	0.21
3	0.42
4	0.32
5	0.41
avg	0.36

(b) Kappa interpretation

Kappa value	Interpretation
<0	Poor agreement
0.01 - 0.20	Slight agreement
0.21 - 0.40	Fair agreement
0.41 - 0.60	Moderate agreement
0.61 - 0.80	Substantial agreement
0.81 - 1.00	Almost perfect agreement

actions at least once, at the same time, and verify the players' actions by watching the replays in-game.

Moreover, we found that the resultant segments and labels are also very inconsistent between annotators. We used Fleiss' Kappa [7] to measure the agreement between them. Table II shows that the two annotators are only in fair agreement with each other. The average Kappa value is 0.36.

Due to the low degree of agreement between our annotators, we decided not to use this labeled data as the ground truths. Two lessons were learned in this effort. Firstly, general players may not be able to reliably segment and label action lists. This leads us to later choose an expert SC2 player who is a postdoc researcher and a Master league player with extensive knowledge of the game to annotate action lists. Secondly, action lists are very long and noisy. The noise filtering rules should be introduced to preprocess the action lists.

3) Rule Based Noise Filtering: Suggested by the expert annotator, we filter out noises from the raw action lists before segmenting and labeling them. Noisy actions are trivial actions that the player performs in the game either accidentally (e.g., wrong keypress or mouse clicks) or purposely (e.g., to inflate her personal gameplay statistics like number of actions per minute). These actions do not affect the game and the game will progress exactly in the same manner if the noise is removed. We worked closely with the expert player to work out the following noise filtering rules:

- *Rule 1: Remove all the non-command actions, such as a 'move screen' action.* The move screen actions only change player's view of the battle field, but do not directly affect the players' strategies.
- *Rule 2: If there are consecutive 'select' actions, keep the last one while removing the rest.* This is because the last selection action supersedes the previous selections.
- *Rule 3: If there are consecutive 'right click' actions, keep the last one while removing others.* Again, the last right click supersedes the previous ones.

We call the filtered action lists the *clean action lists*. The action distribution before and after filtering are shown in Figure 5. We can observe that after filtering noises, the number of action lists with few actions increases significantly. The average number of actions in action list is reduced to 693 from 3517, which means that on average, 80% noise actions in action lists are noises.

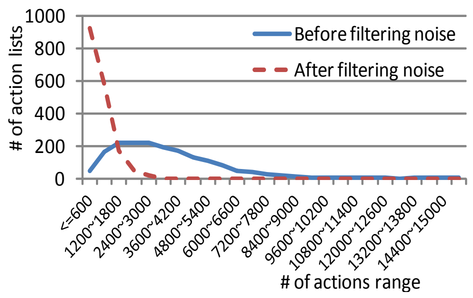


Fig. 5. The action distribution before and after filtering.

4) Expert Annotation: The expert labeled all the clean action lists in our dataset (totally 1732 action lists)[8]. As shown in Table III, there are 10 unique labels defined by the expert labeler. Every segment in the 1732 action lists is assigned one of these 10 labels.

TABLE III
LABELS GIVEN BY THE EXPERT PLAYER

Label name	Description
TRAIN	Training of any combat or worker units
BUILD	Constructing any buildings
UPGRADE	Upgrading units or buildings
RESEARCH	Researching a new technology
USE ABILITY	Using any special abilities or spells
HOTKEY ASSIGN	Assigning hotkeys to units or buildings
MINE	Gathering either minerals or gas
ATTACK	Attack opponent's units or buildings
RALLY POINT	Setting the units' rally point
MOVE	Moving any units or buildings around

IV. CRFs Based Segmentation and Labeling Framework

CRFs are a probabilistic framework which can be applied to segmenting and labeling sequential data. A CRF is a form of undirected graphical model that defines a conditional probability distribution over label sequences given a particular observation sequence. CRFs relax the independence assumptions required by HMMs, which define a joint probability distribution over label sequences given a observation sequence, and also avoid the label bias problem, which is a weakness of MEMMs [9]. For our action list segmentation and labeling problem, the observation sequence is an action list. Since the label of a segment represents the meaning of actions within this segment, we can automatically assign the same label to all the actions in that segment. Therefore, the label sequence in our problem is the list of actions' labels.

[8]http://ink.library.smu.edu.sg/data/1/

Linear-chain CRFs define the probability of a label sequence $\mathbf{y}$ given observation sequence $\mathbf{x}$ to be:

$$p(\mathbf{y}|\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) = \frac{1}{Z(\mathbf{x})}exp(\sum_{i,j}\lambda_j t_j(y_{i-1},y_i,\mathbf{x},i)+\sum_{i,k}\mu_k s_k(y_i,\mathbf{x},i)). \quad (1)$$

where $Z(\mathbf{x})$ is a normalization factor that makes the probability of all label sequences sum to one; $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the parameters to be estimated from training data; $t_j(y_{i-1},y_i,\mathbf{x},i)$ is a transition feature function of the entire observation sequence and the labels at positions i and $i-1$, and λ_j is learned parameter associated with feature t_j; $s_k(y_i,\mathbf{x},i)$ is a state feature function of the label at position i and the observation sequence, and μ_k is learned parameter associated with s_k.

We specify feature functions when the CRF model is to be learned. For example, one boolean transition feature function might be true if action x_{i-1} contains 'select', action x_i contains 'train', label y_{i-1} is 'train', and label y_i is 'train'; one boolean state feature function might be true if the action x_i contains 'build', and label y_i is 'build'.

To simplify our notation, we write

$$s(y_i,\mathbf{x},i) = s(y_{i-1},y_i,\mathbf{x},i) \quad (2)$$

and

$$F_j(\mathbf{y},\mathbf{x}) = \sum_i f_j(y_{i-1},y_i,\mathbf{x},i) \quad (3)$$

where each $f_j(y_{i-1},y_i,\mathbf{x},i)$ is either a state function or a transition function. This allows we simplify the probability as

$$p(\mathbf{y}|\mathbf{x},\boldsymbol{\lambda}) = \frac{1}{Z(\mathbf{x})}exp(\sum_j \lambda_j F_j(\mathbf{y},\mathbf{x},i)). \quad (4)$$

The most probable label sequence for an input $\mathbf{x}$ can be efficiently determined using Viterbi algorithm [15].

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}). \quad (5)$$

Given a training dataset $\{(\mathbf{x}^{(\mathbf{t})},\mathbf{y}^{(\mathbf{t})})\}$, the parameters $\boldsymbol{\lambda}$ in Eq. 4 can be estimated by maximum log-likelihood which is maximizing the conditional probability of $\{\mathbf{y}^{(\mathbf{t})}\}$, given corresponding $\{\mathbf{x}^{(\mathbf{t})}\}$. For a CRF, the log-likelihood is given by:

$$
\begin{aligned}
L(\boldsymbol{\lambda}) &= \sum_t log(p(\{\mathbf{y^{(t)}}\}|\{\mathbf{x^{(t)}}\}, \boldsymbol{\lambda})) \\
&= \sum_t (\sum_j \lambda_j F_j(\mathbf{y^{(t)}}, \mathbf{x^{(t)}}) - logZ_{\mathbf{x^{(t)}}}).
\end{aligned} \quad (6)
$$

The parameters can be identified by using an iterative technique such as traditional method iterative scaling [4], or LBFGS, which is a fast method [3], [10], [16]. More details about CRFs can be found in [9].

A. *Features Extraction*

This section outlines the three types of informative features extracted from action lists. All the features are binary.

- **Current Action Features (Type I):** *These are information extracted from the current action*, such as the current action type (select, right click, attack, upgrade, research, etc.), the current action's target type (worker, soldier, building, resource, etc.), occurrence of a special word (e.g. flying), and player's race (Terran, Zerg and Protoss).
- **Neighboring Action Features (Type II):** *These are information extracted from the neighboring actions of the current action.* For example, the features can be the last action's action type, the 2nd last action's action type, the 3rd last action's action type, the 4th last action's action type, the next action's action type, and so on so forth. Additionally, features in Type II can also be the combination of information from neighboring actions, such as the 2nd last action's target type together with the last action's target type (e.g.: Last Last Target = worker and Last Target = resource).
- **Bigram Features (Type III):** *These are information combined from the current action and the neighbors of the current action*, such as: combine action types of previous and current actions (e.g.: Current = right click and Last = select).

We extract a total of 373,163 binary features. The number of features of the three types of features are listed in Table IV.

TABLE IV
FEATURES USED IN OUR EXPERIMENTS

Type	I	II	III
# of features	46	411	372,706

V. Experiments

In this section, we describe our experiment setup and analyze the performance results.

A. *Experiment Setup*

We first evaluate the action lists segmentation and labeling performance using CRF. The implementation of CRF we used is CRF++ [17], which uses LBFGS to learn the parameters. For comparison, we use Support Vector Machine (SVM) as our baseline method. In our first experiment, we evaluate the performance of CRF and SVM using different number of training action lists including 1, 5, 10, 20, 50, 100, and 200, and the remaining labeled action lists as testing data. We also trained CRF model on 500 and 800 action lists, since their results are very similar to the result of 200 action lists, we will not show them in the figures. In this experiments, we use all features in Table IV to train both CRF and SVM model.

In the second experiment, we show the performance of CRF model using different subsets of features and 200 action lists as training data.

To measure the *overall performance* of segmentation and labeling using CRF and SVM, we use accuracy defined below:

$$
accuracy = \frac{number\ of\ actions\ assigned\ correct\ label}{number\ of\ actions} \quad (7)
$$

To evaluate the *performance for each label*, we used F1-score. Recall $A = \{\alpha_1, \alpha_2, ..., \alpha_h\}$ represents the set of action labels. The F1-score of label α_i is defined as:

$$
F1\text{-}score_{\alpha_i} = \frac{2 \times precision_{\alpha_i} \times recall_{\alpha_i}}{precision_{\alpha_i} + recall_{\alpha_i}} \quad (8)
$$

where

$$
precision_{\alpha_i} = \frac{number\ of\ actions\ assigned\ \alpha_i\ correctly}{number\ of\ actions\ assigned\ \alpha_i} \quad (9)
$$

$$
recall_{\alpha_i} = \frac{number\ of\ actions\ assigned\ \alpha_i\ correctly}{number\ of\ actions\ with\ label\ \alpha_i} \quad (10)
$$

For each training data size involving some number of action lists, say x, we conducted 5 runs of experiments so as to obtain the average accuracy for overall performance and average F1-score for each label performance. Each run used a different set of randomly selected x action lists for training.

B. *Performance Analysis*

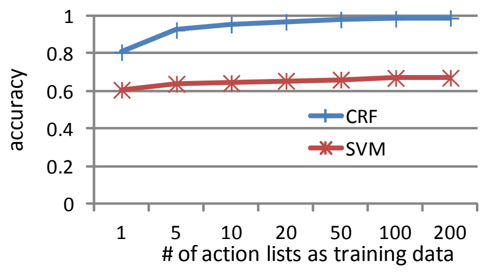
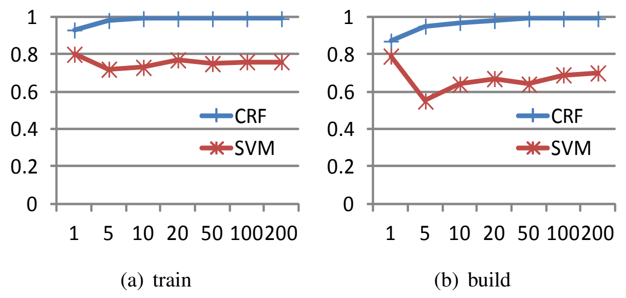
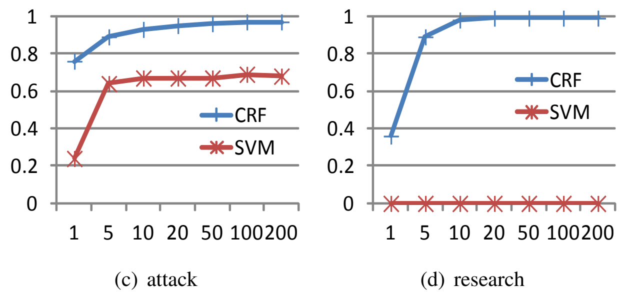
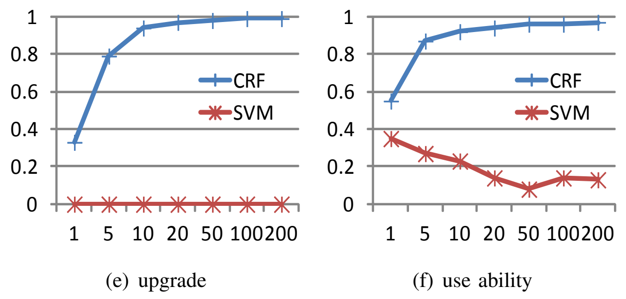


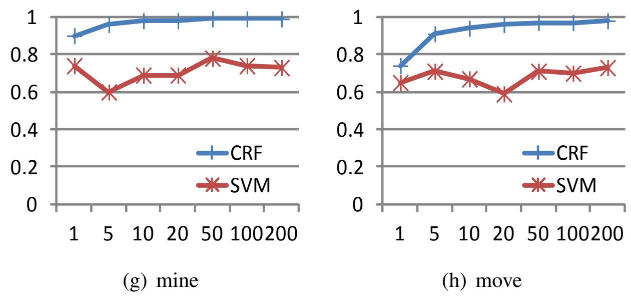
Fig. 6. Accuracy on different training dataset size.

1) Overall Performance: We now present the performance results for CRF and SVM. Figure 6 shows the average accuracy of action list segmentation labeling for different number of training action lists. This figure shows that the performance of CRF is much better than SVM for any training data size.









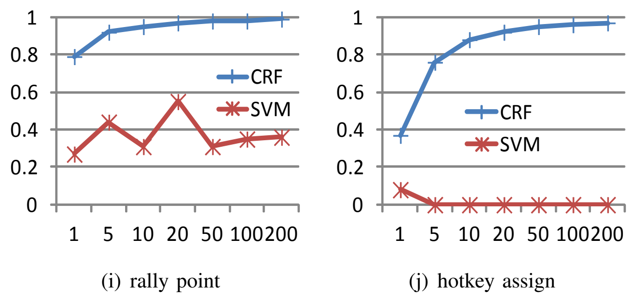


Fig. 7. Performance for different action labels. The x-axes represents number of action lists. The y-axes represents the F1-score.

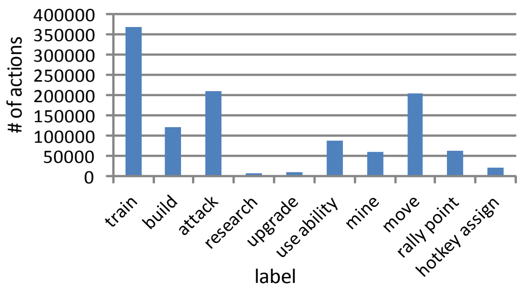


Fig. 8. The number of actions on each label in our whole dataset (1732 action lists).

The performance of CRF improves significantly with more training action lists. However, the performance improvement is less significant for SVM. When we used 200 action lists as training data, the CRF model achieved an accuracy as high as 98.9%.

2) *Performance for Each Label:* We now analyze the performance of CRF and SVM for each action label to gain a better insight. Figure 7 shows the average F1-score of each action label for different number of training action lists.

We observe that the performance of all action labels has a positive correlation with the size of the training data. The results suggest the labels 'train', 'build', 'attack' and 'mine' can be easily segmented and labeled even when few training action lists are given. There are also action labels (e.g. 'upgrade','use ability', etc.) that are more difficult to segment and label. The difficulty can be overcome by having more training action lists. For example, F1-score increases sharply when the number of training action lists increases from 1 to 5. Given sufficiently large number of action lists such as 200, we obtain an F1-score of at least 0.96 for all action labels. Further increase in training data size does not bring about significant improvement.

Figure 7 also shows that SVM performs much worse than CRF for some action labels, particularly for 'research', 'upgrade' and 'hotkey assign' labels. This could be due to imbalanced data in our action lists. Imbalanced data is known to cause poor performance in classification particularly for the minority classes [8], [18]. Figure 8 shows the number of actions with different action labels in our dataset. The number of actions with 'research', 'upgrade', and 'hotkey assign' labels are relatively fewer compared to other labels. It is therefore not a surprise to have SVM performs badly for these classes. Interestingly, CRF appears to be more robust to our imbalanced dataset.

3) *Performance on Different Type of Features:* Figure 9 shows the performance of CRF model using different sets of features including (a) Type I - Current Action Features only, (b) Type II - Neighboring Actions Features only, (c) Type III - Bigram Features only, and (d) All features. This figure shows that using all features gives us the best accuracy. Type I features only gives us the worst performance as compared to other type of features. Compared with Type I features, the

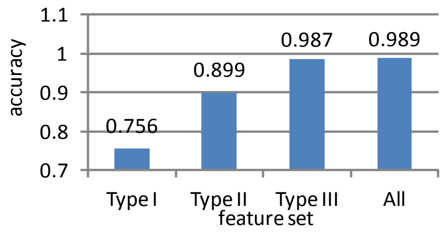


Fig. 9. Accuracy on different subset of features.

performance of using Type II features is better by 14%. This indicates that the neighboring actions can provide more useful information than the current action in determining the action labels. The accuracy of using Type III features is slightly lower than using all features. This result is reasonable as Type III features use the information from both current actions and neighboring actions. On the whole, we conclude that contextual information of an action including the labels of neighboring actions is important for segmenting and labeling action lists.

VI. Conclusion

This paper introduces an action list segmentation and labeling framework to interpret player's actions in their games. In our research, we observe that: (i)the general players segmentation and labeling results are very inconsistent; (ii) on average, 80% of the actions in action lists are noises; (iii) the segments and labels given by our expert player on the filtered action lists are suitable for training a segmentation and labeling model; and (iv)the performance of CRF model for this task is generally quite good. The accuracy of segmentation and labeling increases with training data size, and it could be as high as 98.9%.

Our future work will focus on mining game play strategies from in-game data using action list segments and labels. In RTS game, player's strategy is a plan of actions designed to beat the opponents. It corresponds to how player organizes resources and controls army units. Understanding player strategy is useful for many reasons. Firstly, we can help player understand his/her playing skills at a high level. Secondly, we can help players study the replays of his/her opponents. Thirdly, we can also recommend strategy to players to help them improve their playing skills. Finally, the strategy helps researchers analyze player characters and behaviors.

Acknowledgments

This work is supported by the National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

References

[1] Andrs Belicza. Sc2gears. https://sites.google.com/site/sc2gears/, 2010.

[2] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. "A maximum entropy approach to natural language processing". *Computational Linguistics*, 22:39–71, 1996.

[3] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. "Representations of quasi-newton matrices and their use in limited memory methods". *Math. Program.*, 63:129–156, 1994.

[4] S. Della Pietra, V. Della Pietra, and J. Lafferty. "Inducing features of random fields". *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(4):380–393, 1997.

[5] Ethan W. Dereszynski, Jesse Hostetler, Alan Fern, Thomas G. Dietterich, Thao-Trang Hoang, and Mark Udarbe. "Learning probabilistic behavior models in real-time strategy games". In Vadim Bulitko and Mark O. Riedl, editors, *AIIDE*. The AAAI Press, 2011.

[6] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[7] Joseph L. Fleiss. "Measuring nominal scale agreement among many raters". *Psychological Bulletin*, 76(5):378–382, 1971.

[8] Haibo He and Edwardo A. Garcia. "Learning from Imbalanced Data". *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[9] John Lafferty, Andrew McCallum, and Fernando Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[10] Robert Malouf. "A comparison of algorithms for maximum entropy parameter estimation". In *proceedings of the 6th conference on Natural language learning - Volume 20*, COLING-02, pages 1–7, Stroudsburg, PA, USA, 2002.

[11] Andrew McCallum, Dayne Freitag, and Fernando Pereira. "Maximum entropy markov models for information extraction and segmentation". In *Proc. 17th International Conf. on Machine Learning*, pages 591–598. Morgan Kaufmann, San Francisco, CA, 2000.

[12] Josh McCoy and Michael Mateas. "An integrated agent for playing real-time strategy games". In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, pages 1313–1318. AAAI Press, 2008.

[13] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. "Table extraction using conditional random fields". In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 235–242, New York, NY, USA, 2003. ACM.

[14] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice Hall, united states ed edition, 1993.

[15] L.R. Rabiner. "A tutorial on hidden markov models and selected applications in speech recognition". *Proceedings of the IEEE*, 77(2):257–286, 1989.

[16] Fei Sha and Fernando Pereira. "Shallow parsing with conditional random fields". In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 134–141, Stroudsburg, PA, USA, 2003.

[17] Taku. CRF++: Yet another CRF toolkit. http://crfpp.sourceforge.net/, 2008.

[18] Jason Van Hulse, Taghi M. Khoshgoftaar, and Amri Napolitano. "Experimental perspectives on learning from imbalanced data". In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 935–942, New York, NY, USA, 2007.

[19] Lauri Virkamaki. Phpsc2replay: SC2 replay parser for PHP. http://code.google.com/p/phpsc2replay/, 2010.

[20] Ben G. Weber and Michael Mateas. "A data mining approach to strategy prediction". pages 140–147, 2009.

[21] Ben G. Weber and Michael Mateas. "Case-based reasoning for build order in real-time strategy games. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2009)*, 2009.

[22] Ben G. Weber, Michael Mateas, and Arnav Jhala. "A particle model for state estimation in real-time strategy games". In *Proceedings of AIIDE*, pages 103–108, Stanford, Palo Alto, California, 2011.