Predictive Analysis of Real-Time Strategy Games Using Discriminative Subgraph Mining

Jennifer L. Leopold[1], Isam A. Alobaidi[1], and Nathan W. Eloe[2]

[1]Missouri University of Science & Technology
Department of Computer Science
Rolla, MO, USA
[2]Northwest Missouri State University
School of Computer Science and
Information Systems
Maryville, MO, USA
leopoldj@mst.edu, iaahgb@mst.edu, nathane@nwmissouri.edu

Abstract. Real-Time Strategy (RTS) video games are not only a popular entertainment medium, they also are an abstraction of many real-world applications where the aim is to increase your resources and decrease those of your opponent. An obvious application is a military battle; yet another example is a person's physical health where it is advantageous to increase the number of healthy cells in the body and destroy cancerous cells (wherein cancer is your opponent). Using predictive analytics, which examines past examples of success and failure, we can learn how to predict positive outcomes for such scenarios. Herein we show how discriminative subgraph mining can be employed to analyze a collection of played RTS games, and make recommendations about sequences of actions that should, as well as should not, be made to increase the chances of winning future games. As proof of concept, we present the results of an experiment that utilizes our strategy for one particular RTS game.

Keywords: Graph mining, Game Mining, Predictive Analytics.

1 Introduction

Real-Time Strategy (RTS) games are a subgenre of strategy video games wherein the participants position and maneuver units (e.g., troops, robots, and drones) and structures under their control to secure areas and destroy their opponent's assets. In some games, the created entities can in turn create and destroy other entities. Hence the focal points of such games are: resource generation and destruction, and indirect control of units and structures (via other units and structures). RTS games typically have a diverse set of resources which the player can deploy, basically offensive or defensive in nature, and a large variety of

environments/storylines from which to select, often with a military science fiction theme; a popular and sophisticated example is StarCraft. The games are usually multi-player, with the winner determined by some criterion such as the player with the most assets at the end of a certain time period or by the last player remaining after all other players' assets have been depleted. Although the RTS game scenario is used for entertainment purposes, it can be abstracted as a model for real-world applications such as military battles, cyberinfrastructure networks that may need to be managed as they come under malicious attack, and even disease history/diagnosis systems which track a patient's symptoms, treatments, and disease progression over time.

Herein we test the hypothesis that predictive analytics can be employed to examine a collection of played games and make recommendations as to what a player should do and what a player should not do in order to increase the chances of winning the game the next time s/he plays. As proof of concept, this method will be tested for one particular RTS game; however, the method that we employ should be applicable to any multi-player RTS game and possibly could be generalized to sequences of categorically offensive versus defensive moves for any RTS game. Specifically, we will model the moves of each played game as directed graphs for the winner's and loser's moves, respectively, and apply discriminative subgraph mining to identify our game strategy recommendations.

The organization of this paper is as follows. Section 2 provides a brief overview of game data mining, data mining techniques used in predictive analytics, and discriminative subgraph mining. Section 3 explains the discriminative subgraph mining algorithm that we utilized for our study. Section 4 outlines the experiment that we conducted to test our hypothesis and the results that we obtained. A summary and conclusions of our research are given in Section 5. Future work is discussed in Section 6.

2 Related Work

2.1 Game Data Mining

For years there has been interest in analyzing games played by others in order to become a more competitive player. In its earliest form, people sought to identify the moves in the game that led to desirable, rather than undesirable, outcomes. For many games it is not only the quantity of assets, but particular features of the assets in the game that must be considered (e.g., an asset's functionality and location). For example, in the game of chess, given the choice, it is usually better to have one bishop than three pawns; position of a piece on the game board is also important as a bishop that is blocked by other pieces may not be able to attack. A number of studies have been conducted wherein a database of played games is analyzed to determine the winning percentage under various scenarios such as games in which one player has two bishops and no knights and the other player has two knights and no bishops after some point in a chess game; see [1, 2] for examples of such studies.

Contemporary genres of games, such as RTS video games, have a much more sophisticated collection of assets (e.g., game pieces) than traditional games such as chess and the characteristics of the assets can be much more diverse. Accordingly, analysis of desirable asset acquisition and deployment throughout a game has become more complex and computationally expensive.

Another branch of game data mining, also known as game telemetry, involves analysis of the people who play the game and/or the personas they may create. There are databases of this information for various online games and mining software to analyze data such as the players' skill level and time spent having played the game; see [3] for an example of such software. Some analyses may try to relate features from a player's profile to his/her winning percentage and odds of winning future games. This area of study is not the focus of the research pursued herein; we do not consider any data related to a player's profile.

As is discussed in [4], the intentions of game data mining should be made clear. *Description* describes patterns found in the game data; similarly, *characterization* is a summation of some general features associated with the data. These patterns could be independent of whether they occurred in the winners' games or the losers' games, or whether the patterns occurred in a majority or a minority of the games in the dataset. Description and characterization are the fundamental, general goals of most data mining efforts. *Classification* (and *clustering*) are used to compare and organize some features of the data into classes; with game data this usually isn't necessary since we are most interested in classifications as winning and losing games, information which is already known. *Discrimination* seeks to identify the differences between groups of instances in the game data beyond just the classification of winning and losing. *Prediction* has the goal of providing a rule (or some form of guideline) that can be used as guidance for playing or forecasting the outcome of future games. The work presented in this study focuses on discrimination and prediction of game data.

2.2 Data Mining Techniques Used in Predictive Analytics

Utilizing mathematical modeling, the field of predictive analytics examines past examples of success and failure to determine the variables that lead to successful outcomes and can be used to make predictions about future events. It has been used widely in the financial and insurance sectors. Here we briefly discuss some of the most common types of data mining methods used for predictive analytics.

Regression analysis: Linear regression is one type of regression analysis commonly used for predictive analytics. This method analyzes the relationship between a dependent variable and a set of independent variables. The relationship is expressed as an equation that predicts the value of the dependent variable as a linear function of the independent variables. For game data the dependent variable is typically the outcome of the game (i.e., win or lose) and the independent variables can be the various possible moves. Given the number of possible

moves in an RTS game and the number of possible sequences of moves, this method could be computationally prohibitive.

Rule induction: Rule induction methods such as association rule mining seek to find relationships between variables in the dataset. For example, it could be determined that when the player does actions A and B, s/he also does actions C and D. By applying association rule mining on only the winners' games, we could identify some actions that winning players did. Similarly, by mining the losers' games, we could find some actions common to losing players. However, we then would have to examine the differences between those rule sets to gain knowledge about what winners did that losers did not do, and vice-versa. It should be noted that rule mining typically generates a considerable number of rules because of its combinatorial approach; typically, only rules meeting a certain support threshold are retained.

Decision trees: Decision trees are most often used for classification and can be thought of as a graphical depiction of a rule; each branch of a decision tree can be thought of as a separate rule consisting of a conjunction of the attribute predicates of nodes along that branch. One approach would be to construct decision trees from the winning games and losing games, respectively. Resultingly, the issues previously mentioned for association rule mining of the RTS game data would apply for decision tree methods as well.

Clustering: Clustering is a way to categorize a collection of instances in order to look for patterns; groups are formed to maximize similarity between the instances within a group and to maximize dissimilarly between instances in different groups. Game data are already clustered into two groups: winners and losers. For the purpose of analyzing successful (and unsuccessful) actions, we would likely attempt to form clusters of action sequences. As with linear regression, given the number of possible moves in an RTS game and the number of possible sequences of moves, this method would be computationally prohibitive, and likely would result in an uninformative number of clusters, unless some type of feature reduction mapping method was employed (i.e., mapping specific actions and their time of occurrence in the game to more generalized representations).

Neural networks: Neural networks are composed of a series of interconnected nodes that map a set of inputs into one or more outputs. The interconnections between inputs (which, for the game data, could be actions in the game) are determined based on an analysis of the played games. As with clustering, this method likely would be computationally prohibitive, and would probably not yield useful results, for the RTS game data unless we employed some type of data reduction mapping, which subsequently could result in loss of useful, specific information.

2.3 Discriminative Subgraph Mining

Many problems can be modeled with graphs, wherein entities are represented as vertices and relationships between entities are represented as edges. When the relationship between two vertices has some semantic distinction of a predecessor and a successor, the edges are

directed and hence the graph is considered directed. A played RTS game can be modeled as a directed graph where each action (e.g., move) is represented by a vertex and an edge represents two consecutive actions that were made in the game. By necessity, each vertex also must be identified by which player performed that action. The moves for one player do not form a strictly linear sequence because an action can generate multiple actions; for example, the player may create a drone which in turn simultaneously spawns 5 more drones, each of which becomes a new vertex, and 5 edges are created from the propagating drone vertex.

Finding interesting patterns in graphs (both directed and undirected) has been well-researched and is applicable to many problem domains in fields such as bioinformatics, cheminformatics, and communication networks. An 'interesting' pattern in a graph could be a subgraph that appears frequently over a collection of graphs or could be a subgraph that has a particular topography (e.g., a clique). Another type of interesting pattern is a discriminative subgraph.

Discriminative subgraph mining seeks to find a subgraph that appears in one collection of graphs, but does not appear in another collection of graphs. This approach has been used to study several problems including identifying chemical functional groups that trigger side-effects in drugs [5], classifying proteins by amino acid sequence [6], and identifying bugs in software [7, 8, 9]. Here we briefly discuss some of the strategies that have been employed for discriminative subgraph mining.

In [10] the authors define global-state networks, a collection of graphs that represent a series of snapshots taken over a period of time and model some event. Each snapshot graph has the same topology, but the nodes and/or edges in each graph may have different values. The authors' technique, MINDS, is specifically designed to find minimally discriminative subgraphs in large global-state networks. The network graph search space is organized as a set of decision trees to scrutinize the changes from one snapshot to the next in the collection. To reduce an exponential subgraph search space, they employ a Monte Carlo Markov sampling strategy. While the strategies employed in MINDS were found to work well for the global-state networks, they would not be appropriate for the RTS game dataset where each game, and hence each graph's topology, can differ significantly from other games. Additionally, as will be discussed in Section 3, game data mining should not necessarily be limited to just finding minimally discriminative subgraphs.

Discriminative subgraph mining was used in [11] to find subgraphs that would cover as many positive examples and as few negative examples as possible. The test dataset contained protein structures possessing a specific function and proteins not having that function. Each graph contained approximately 1,000 edges and was very dense (i.e., in terms of the number of edges relative to the number of vertices in the graph). Two heuristics were employed to reduce the computational complexity of the mining process. Together these heuristics were used to assign a score to each candidate discriminative subgraph; the score considered the number of positive graphs minus the number of negative graphs in which the subgraph was found. Only the smallest such subgraphs with high scores were returned in the results; any

(larger) subgraph that contained one of these (smaller) subgraphs was not further examined, thereby reducing the search space. This algorithm could have been adapted for the predictive game strategy study, but would have had to have been run for both the cases of the winning games being the positive examples and the losing games being the negative examples, and the winning games being the negative examples and the losing games being the positive examples in order to find recommendations for what should and should not be done to win the game.

Another strategy for dealing with the large search space normally incurred with discriminative subgraph mining was presented in [12]. As discussed above, a scoring scheme was used to evaluate the discrimination potential of candidate subgraphs. However, The Learning To Search (LTS) algorithm of [12] differed from the work of [11] by combining the scoring scheme with a sampling strategy to select candidate subgraphs. Candidates deemed promising (in terms of their score) were added to a list and further extended with edges for additional consideration; non-promising candidates were discarded, thereby implementing a branch-and-bound search. This method was tested on protein datasets with good prediction accuracy and a faster runtime than some other discriminative mining methods. As with the algorithm in [12], this approach possibly could be used to analyze a strategy game dataset.

Discriminative subgraph mining also has been used to find bugs in software in [7, 8, 9]. For this application, a program is modeled as a graph based on its control flow graph. In brief, a control flow graph is a directed graph made up of nodes representing basic blocks. Each basic block contains one or more statements from the program. There is an edge from basic block Bi to basic block Bj if program execution can flow from Bi to Bj. Traces through the control flow graph for inputs that produce correct results forms one collection of graphs and traces for inputs that produce incorrect results forms a second collection of graphs. The idea is to look for a discriminative subgraph between the two collections of graphs; this represents the lines of code that are, or are not, being executed when the bug occurs. The algorithm presented in [7] utilizes the LEAP algorithm [13] as a branch-and-bound heuristic on the search space of graphs that it examines; it is based on the observation that subgraphs with higher frequency are more likely to be discriminative. This algorithm was modified slightly to specifically scrutinize certain programming constructs and subsequently was tested in [8, 9]. This general approach to discriminative subgraph mining is applicable to the RTS game dataset and is discussed in more detail in the next section.

3 Methodology: Discriminative Subgraph Mining

The algorithm we employed for discriminative subgraph mining is similar to the approach taken in [8, 9], but does not employ any heuristics specific to game data. Although we ran it sequentially, it easily lends itself to parallel or distributed processing.

Let C^+ and C^- represent two sets of (undirected or directed) graphs for which we want to find a discriminative subgraph; that is, we want to find a subgraph that appears in the graphs in C^- and does not appear in the graphs in C^+, or vice-versa. We shall refer to C^+ as the positive graphs and C^- as the negative graphs although this naming convention has no direct semantic correlation to the classification of the graphs in those respective sets (e.g., 'winner' does not necessarily mean positive). The function *FindDiscriminativeGraph* (Fig. 1) first removes non-discriminative edges from the graphs in both sets; since such edges appear in the graphs in both sets, they cannot be used to differentiate the graphs in the those sets. *FindDiscriminativeGraph* then calls *CreateDiscriminativeGraph* (Fig. 2) to try to find a subgraph that is common to all graphs in C^-, but not common to all the graphs in C^+. If we are unable to find such a graph, then the function *RelaxedCreate-DiscriminativeGraph* (Fig. 3) is called, which relaxes the requirement that the subgraph we seek not be present in all of the C^+ graphs; instead the subgraph only has to not be present in $\alpha * |C^+|$ of the C^+ graphs, where α is a user-specified parameter (our default is $\alpha = 0.5$).

FindDiscriminativeGraph and *CreateDiscriminativeGraph* use a function called *Augment*; this function takes a subgraph G and adds to it an edge (and possibly a node) such that the source vertex exists in G, and the edge (and destination node) exists in all graphs in subgraph collection S1. In this way, a subgraph with an additional edge that exists in all elements of S1 is created and considered by the algorithm.

If we still fail to find a discriminative subgraph, then the difference likely does not involve edges that are in all graphs in C^- and not in graphs in C^+, but rather involves edges in the C^+ graphs that are not in the C^- graphs. Thus, we again call *CreateDiscriminativeGraph*, but reverse the order of the parameters (C^+ and C^-) from our previous call. If we still fail to find a discriminative subgraph, we again call *Relaxed-CreateDiscriminativeGraph* and look for a subgraph that only has to not be present in $\beta * |C^+|$ of the C^+ graphs, where β is a user-specified parameter (our default is $\beta = 0.5$).

It is possible that the resulting discriminative graph will be disconnected. Additionally, it could be the case that multiple subgraphs could qualify as a discriminative subgraph. The algorithm addresses both of these cases by returning the maximal discriminative subgraph; this result may be disconnected and will include all possible discriminative edges. It should be noted that it also is possible that our algorithm will not find any subgraph that meets the discriminative conditions. This could occur if the requirement that at least α (β) of the graphs in C^- (C^+) must have at least one edge in common has not been satisfied.

The computational complexity of the process is dependent upon the number of graphs in each collection and the number of edges in each graph. As specified in line 1 of *CreateDiscriminativeGraph*, we begin by examining each single edge from each graph in one of the graph collections. However, in lines 7-9 of that algorithm, we potentially build larger subgraphs that must be searched for; this is the subgraph isomorphism problem, which is NP-complete.

Algorithm: *FindDiscriminativeGraph(C+, C-, α, β)*
C+: set of positive graphs
C-: set of negative graphs
α: percentage of graphs that discriminative subgraph need not be present in C+ when relaxing conditions
β: percentage of graphs that discriminative subgraph need not be present in C- when relaxing conditions

```
1.  remove non-discriminative edges from graphs in C+ and C-;
2.  G = CreateDiscriminativeGraph(C-, C+);
3.  if G is empty then
4.    G = RelaxedCreateDiscriminativeGraph(C-, C+,
                                           |C+| * α);
5.    if G is empty then
6.      G = CreateDiscriminativeGraph(C+, C-);
7.      if G is empty then
8.        G = RelaxedCreateDiscriminativeGraph(C+, C-,
                                               |C-| * β);
9.      end-if;
10.   end-if;
11. end-if;
12. return G
```

Fig. 1. Algorithm for *FindDiscriminativeGraph*

Algorithm: *CreateDiscriminativeGraph(S1, S2)*
S1: set of graphs
S2: set of graphs

```
1.  FreqSG = queue of 1-edge subgraphs in S1;
2.  while FreqSG is not empty do
3.    G = FreqSG.dequeue();
4.    if G is not in any graph in S2 then
5.      return(G);
6.    end-if;
7.    NewGraphs = Augment(G);
8.    for each graph G' in NewGraphs do
9.      FreqSG.enqueue(G');
10.   end-for;
11. end-while;
12. return(empty graph)
```

Fig. 2. Algorithm for *CreateDiscriminativeGraph*

```
Algorithm: RelaxedCreateDiscriminativeGraph(S1, S2, γ)
S1: set of graphs
S2: set of graphs
γ: threshold for number of graphs discriminative subgraph must be present in
1.  FreqSG = queue of 1-edge subgraphs in S1;
2.  while FregSG is not empty do
3.     G = FreqSG.dequeue();
4.     if G is in < γ graphs in S2 then
5.        return(G);
6.     end-if;
7.     NewGraphs = Augment(G);
8.     for each graph G' in NewGraphs do
9.        FreqSG.enqueue(G');
10.    end-for;
11. end-while;
12. return(empty graph)
```

Fig. 3. Algorithm for *RelaxedCreateDiscriminativeGraph*

4 Experiment and Results

In this section we discuss the details of an experiment we conducted to test the hypothesis that predictive analytics, specifically discriminative subgraph mining, can be employed to examine a collection of played strategy games and make recommendations as to what a player should do, and should not do, in order to increase the chances of winning the game in the future.

4.1 Experimental Setup

The game that we selected is an online, multi-player RTS game called Interloper [14]. Interloper was chosen over more sophisticated RTS games like StarCraft because of its relatively limited set of action types which include: creating territory tiles, spawning drones, spawning blockades, creating units (e.g., sentinels, drones, defenders, destroyers, markers, bombs, blockades, and snipers), building structures, destroying targets, moving and positioning characters, removing characters, hitting characters, and exploding characters. We obtained a database of 19 played Interloper games from the game's developer. Each of these games contained the sequence of actions performed by each of two players, with a designation of which player won the game. Each action type in the data file had a documented integer encoding. The total number of moves (for both players) in a game in the dataset ranged from 183 to 5,338.

For each game in the dataset we created two individual files: one for the winner's moves and one for the loser's moves. The format for each of the data files that we created was

modeled as a directed graph, one edge per line, where each vertex was an action, and an edge represented a consecutive sequence of (two) actions made in that game. As with games such as chess, we thought it would be interesting to analyze (and make recommendations for) the game in three phases: the beginning game, the middle game, and the end game. In chess there is no clear definition of when the middle game begins and ends, or when the end game begins. Similarly, we had no such guidelines for Interloper. Therefore, we simply divided each game file into the first third number of moves, the middle third number of moves, and the last third number of moves, and referred to these as phases 1, 2, and 3 of the games, respectively. Each phase was analyzed separately.

As described in the previous section, our discriminative subgraph mining algorithm would not find a discriminative subgraph unless a certain percentage of the graphs in each (C^- or C^+) "collection" had at least a certain percentage of edges in common. Therefore, we had to test small groups of games at a time. To make sure that we did not miss any possible common edges, we tested every combination of two winning and two losing graphs; that is, a pair of winning graphs played the role of C^+ in *FindDiscriminativeGraph* and a pair of losing graphs played the role of C^-. We then reversed the roles (i.e., a pair of losing graphs played the role of C^+ and a pair of winning graphs played the role of C^-). Depending on whether the discriminative subgraph was found in C^+ or C^- for the particular assignment to those parameters told us whether the moves should be recommended as something that should be done in order to increase the chance of winning (because it was a difference found in the winning graphs) or something that should not be done (because it was a difference found in the losing graphs).

To test the predictive accuracy of our method, we performed cross validation on the dataset of 19 played games. For phase 1, we used 5-fold cross validation. Five partitions were created, 4 of which contained 4 games and 1 of which contained 3 games; by 'game' we mean both the winner and loser for that game. A random number generator (www.random.org/lists/) was used to determine which games were assigned to each partition (with no duplication). For each of the 5 iterations of the 5-fold cross validation, the "training" dataset was formed from 4 of the partitions and the "test" dataset was the remaining partition; the roles of the partitions were rotated through each iteration of the 5-fold cross validation. Discriminative subgraphs were determined from all possible pairs of winning and losing games in the "training" dataset (i.e., 4 of the 5 partitions). This resulted in a set of subgraphs that formed the recommendations for actions that should be done and a set of subgraphs which formed the recommendations for actions that should not be done in order to win the game.

The error rate was calculated as follows. If a recommendation for what should be done (subgraph) was found in one of the winning graphs in the test partition, it was counted as a true positive (TP); if instead that recommendation (subgraph) was found in one of the losing graphs in the test partition, it was counted as a false positive (FP). If a recommendation for what should not be done (subgraph) was found in one of the losing graphs in the test partition, it was counted as a true negative (TN); if instead that recommendation (subgraph)

was found in one of the winning graphs in the test partition, it was counted as a false negative. The error rate was calculated as 1 – ((TP + TN) / (TP + TN + FP + FN)), and was averaged over the five iterations of the 5-fold cross validation.

For phases 2 and 3 of the game, significantly fewer discriminative subgraphs were found than for phase 1; this will be discussed in the next section. Therefore, instead of creating 5 partitions for cross-validation, we only created 3 partitions: 2 partitions contained 6 games and 1 partition contained 7 games. Consequently, only 3 iterations were run in those cross validations instead of 5. As was done for phase 1, games still were randomly chosen for each partition for each test. All cross-validation tests (for all phases) were repeated 5 times.

It should be noted that the discriminative subgraph mining algorithm was implemented in Python 3.7. A combination of Python programs and bash scripts were created for data file conversions and batch program executions. All programs were executed on a Dell Intel i7-7700 3.60 GHz 64 GB RAM Windows 10 PC.

4.2 Experimental Results

Each of the three phases of the game was analyzed separately using cross-validation, with each cross-validation test repeated 5 times with randomized data (game) assignment for training and test data from the 19-game dataset. Table 1 shows the average error rate for each of the cross-validation tests for each phase, as well as the average error rate over each phase's 5 tests. The resulting predictive accuracy was good, considering that, in general, discriminative subgraphs can have very low frequencies. The collective recommendations (for moves that should be made and moves that should not be made) were accurate approximately 86.5%, 92.4%, and 98.7% of the time for phases 1, 2, and 3 of the game, respectively. It should be noted that the accuracy for phases 2 and 3 were likely much higher than for phase 1 in part because 3-fold (rather than 5-fold) cross validation testing was used for those phases and because there were significantly fewer discriminative subgraphs to test in those phases.

Table 1. Cross-Validation Test Results

Test No.	Phase 1 Avg. Error Rate	Phase 2 Avg. Error Rate	Phase 3 Avg. Error Rate
1	14.40%	10.60%	1.00%
2	13.40%	0.08%	1.60%
3	13.00%	9.10%	0.70%
4	13.50%	9.80%	2.00%
5	13.30%	8.50%	1.40%
Avg.	**13.52%**	**7.62%**	**1.34%**

For phase 1 of the game, when testing all pairs of 2 winning and 2 losing graphs, 2,333 discriminative subgraphs were found that constituted "should do" recommendations and 2,270 discriminative subgraphs were found that represented "should not do" recommendations. The average size of the "should do" recommendation subgraphs was 28 edges; the smallest had 1 edge and the largest had 170 edges. The average size of the "should not do" recommendation subgraphs was 22 edges; the smallest had 1 edge and the largest had 168 edges.

Of the ten most frequently recommended "should do" subgraphs, 3 contained 3 edges (i.e., 4 moves) and 4 contained 4-5 edges (i.e., 5-6 moves). In contrast, 5 of the 10 most frequently recommended "should not do" subgraphs contained only 1 edge (i.e., 2 moves) and 5 contained 2-3 edges (i.e., 3-4 moves). Thus, for this phase of the game, we are not able to provide quite as much information about what a player should not do as we can say about what a player should do.

The types of actions in the phase 1 discriminative subgraphs were predominantly only two types: creation of territory tiles and (fast) moves of a game character. In the Interloper game, creation of territory files can be considered an offensive action against one's opponent. Movement of a game character could be either an offensive or defensive action; the player's intent (e.g., moving away from danger versus moving to a more strategic position in the game space) cannot be deduced from the game data. Another observation that can be made from these particular discriminative subgraphs is a counter that is associated with both of these types of moves. For each game, the counter for each type of action begins at 1 and is incremented by 1 each time that type of action occurs. The phase 1 discriminative subgraphs differed not only in sequences of territory tile creation and character movement, but also in how relatively early (or late) those actions occurred and in what succession. For example, an edge (2800029, 2800030) represents two tile creations with counters 29 and 30, indicating that these were tile creations that occurred well after the game had started (i.e., they were the 29^{th} and 30^{th} tile creations that this player made). Their occurrence in a discriminative subgraph would indicate that it either is or is not advisable to create so many tiles (back to back) in the first phase of the game.

For phase 2 of the game, when testing all pairs of 2 winning and 2 losing graphs, 250 discriminative subgraphs were found that represented "should do" recommendations and 213 discriminative subgraphs were found that characterized "should not do" recommendations. These were about 90% less than the respective numbers of subgraphs found in phase 1. This is not surprising as the number (and order) of different moves that a player could (and likely did) make increased at this point in the game, thereby reducing the number of graphs that had edges in common and could meet the criteria of *FindDiscriminativeGraph*. The average size of the "should do" recommendation subgraphs was 25 edges; the smallest had 1 edge and the largest had 274 edges. The average size of the "should not do" recommendation subgraphs was 14 edges; the smallest had 1 edge and the largest had 155 edges.

The most frequently recommended "should not do" subgraphs in phase 2 only contained a single edge (i.e., 2 moves); thus, there was a further decrease in the amount of information we could provide a player in terms of what not to do in order to win the game. In contrast, 3 of the top 6 most frequently recommended "should do" subgraphs contained at least 11 edges (i.e., 12 moves). Overall, compared to phase 1, this can be seen as the ability to provide much more information about what a player should do in order to win the game during this phase. Unfortunately, again the types of actions that occurred in the discriminative subgraphs were limited, mostly moving a game character (although now at a slower speed than in phase 1); we had anticipated seeing more offensive actions during this phase of the game.

For the final phase of the game, 68 discriminative subgraphs were found that characterized "should do" recommendations; this was a 97% decrease from the number found in phase 1 and a 72% decrease from the number found in phase 2. In this phase, 36 discriminative subgraphs were found that represented "should not do" recommendations; this was a 98.4% decrease from the number of such subgraphs found in phase 1 and an 83% decrease from the number found in phase 2. As mentioned previously, the moves in this phase of the game likely varied more from game to game, and, as such, it became more difficult to meet the criteria stipulated in *FindDiscriminativeGraph*. The average size of the "should do" recommendation subgraphs was 22 edges, which was only slightly smaller than what had been seen in the other two phases; the smallest had 1 edge and the largest had 115 edges, which was by far the smallest of the three phases. The average size of the "should not do" recommendation subgraphs was 18 edges, which is the average size between what was seen for phases 1 and 2; the smallest had 1 edge and the largest had 145 edges, which was slightly smaller than in phase 2. There were 92% fewer discriminative subgraphs found in phase 3 than had been found in phase 1.

For phase 3, we finally saw some of the most frequently recommended "should not do" subgraphs have multiple edges (i.e., more than 2 moves); of the top 7 such subgraphs, 3 contained more than 4 edges, and 2 of those contained 9-10 edges. Amongst the top 7 most frequently recommended "should do" subgraphs, only 1 had a single edge; the average number of edges for the others in this list was 7 edges (i.e., 8 moves). Although we could provide more recommendations about what 'not to do' in phase 3 than for phases 1 and 2, we still could provide much more information about what 'to do' during this phase of the game.

The majority of the actions in the "should do" subgraphs still involved (slow) movement of a game character whereas the actions in the "should not do" subgraphs predominantly involved territory tile creation, removal of a game character, and/or positioning of a game character. Territory tile creation and removal of a game character can be considered offensive actions in Interloper; as mentioned previously, positioning of a game character could be for offensive or defensive purposes, which cannot be determined from the game data. We were surprised that none of the discriminative subgraphs (for any of the phases)

included any defensive actions (e.g., spawning a blockade); however, there are by far more offensive types of actions in the game than defensive actions.

Of all the pairs of 2 winner and 2 loser graphs tested, only a few failed to produce a discriminative subgraph. There were no contradictory results; that is, it was never the case that a sequence of actions in essence would be both recommended and not recommended. Some test pairs produced the same results as other pairs; duplicates were not included in the counts of discriminative subgraphs reported for each phase. Some test pairs produced discriminative subgraphs that were subgraphs of other reported discriminative subgraphs; this was not unexpected since some test (game) pairs had edges in common.

5 Summary and Conclusions

Herein we tested the hypothesis that a form of predictive analytics, namely discriminative subgraph mining, can be used to examine a set of played strategy games and generate a set of recommendations that could be used to predict the chances of winning the game in the future. Using a dataset of played games of a multi-player, Real-Time Strategy (RTS) video game, Interloper, we modeled each game as a graph and found a collection of subgraphs that specified sequences of actions that players should, and should not, make in each of three phases of the game. Although the dataset only contained 19 games, the experimental results showed that the accuracy of our recommendations was high. Overall, our recommendations for our test game, Interloper, were more informative in terms of what a player should do at each of three phases of the game in order to win; however, we also were able to provide some information about what the player should not do. Most importantly, this study has served as a proof of concept that this approach may be a promising strategy for not only game predictive analytics, but also for other problem domains that involve direct and indirect resource generation and destruction.

6 Future Work

We plan to test our discriminative subgraph mining approach on other types of RTS games. If we have the success that we had with Interloper, we hope to establish a mapping between action types and assets in this genre of games so that a more generalized recommendation system can be developed. We also hope to explore ways to make the algorithms more efficient, perhaps applying some heuristics to reduce the search space that are inherent to the nature of game data. Ultimately, we intend to abstract this strategy to other problem domains such as a health care disease tracking and prediction systems using the same foundation of analyzing examples of success and failure in order to make recommendations for future positive outcomes.

References

1. Kaufman, L.: "The Relative Value of the Pieces", Computer Chess Reports, 4:2, pp. 33-34 (1994).
2. Sturman, M.: "Beware the Bishop Pair", Computer Chess Reports, 5:2, pp. 58-59 (1995).
3. Braun, P., Cuzzocrea, A., Keding, T., Leung, C., Padzor, A., and Sayson, D.: "Dame Data Mining: Clustering and Visualization of Online Game Data in Cyber-Physical Worlds", International Conference on Knowledge Based and Intelligent Information and Engineering Systems, pp. 2259-2268, Marseille, France (2017).
4. Drachen, A., Thurau, C., Togelius, J., Yannakakis, G., and Bauckhage, C.: Game Data Mining, Springer-Verlag, Heidelberg, Germany (2011).
5. Shao, Z., Hirayama, Y., Yamanishi, Y., and Saigo, H.: "Mining Discriminative Patterns from Graph Data with Multiple Labels and Its Application to Quantitative Structure-Activity (QSAR) Models", Journal of Chemical Information Models, 55(12), pp. 2519-2527 (Dec. 2015).
6. Jin, N. and Wang, W.: "Discriminative Subgraph Mining for Protein Classification", Computational Knowledge Discovery for Bioinformatics Research, pp. 279-295 (2012).
7. Cheng, H., Lo, D., Zhou, Y., Wang, X., and Yan, X.: "Identifying Bug Signatures Using Discriminative Graph Mining", ISSTA, pp. 141-151, Chicago, IL (2009).
8. Leopold, J., Eloe, N., and Taylor, P.: "BugHint: A Visual Debugger Based on Graph Mining", Proceedings of the 24th International Conference on Visualization and Visual Languages, pp. 109-118, San Francisco, CA (2018).
9. Leopold, J., Eloe, N., Gould, J., and Willard, E.: "A Visual Debugging Aid Based on Discriminative Graph Mining", Journal of Visual Languages and Computing, (to appear February 2019).
10. Ranu, S., Hoang, M., and Singh, A.: "Mining Subgraphs from Global-State Networks", KDD, pp. 509- 517, Chicago, IL (2013).
11. Fuksova, A., Kuzelka, O., and Szaboova, A.: "A Method for Mining Discriminative Graph Patterns", Proceedings of NIPS Machine Learning in Computational Biology Workshop (2013).
12. Jin, N. and Wang, W.: "LTS: Discriminative Subgraph Mining by Learning from Search History", Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 207-218, Hannover, Germany (2011).
13. Yan, X., Cheng, H., Han, J., and Yu, P.: "Mining Significant Graph Patterns by Leap Search", SIGMOD 2008, pp. 433-444, Vancouver, BC, Canada (2008).
14. Interloper Game, http://interlopergame.com, last accessed 2019-12-01.