

Identifying Patterns in Combat that are Predictive of Success in MOBA Games

Pu Yang
North Carolina State
University
Raleigh, NC 27695-8206
pyang3@ncsu.edu

Brent Harrison
North Carolina State
University
Raleigh, NC 27695-8206
beharri5@ncsu.edu

David L. Roberts
North Carolina State
University
Raleigh, NC 27695-8206
robertsd@csc.ncsu.edu

ABSTRACT

Multiplayer Online Battle Arena (MOBA) games rely primarily on combat to determine the ultimate outcome of the game. Combat in these types of games is highly-dynamic and can be difficult for novice players to learn. Typically, mastery of combat requires that players obtain expert knowledge through practice, which can be difficult to concisely describe. In this paper, we present a data-driven approach for discovering patterns in combat tactics that are common among winning teams in MOBA games. We model combat as a sequence of graphs and extract patterns that predict successful outcomes not just of combat, but of the entire game. To identify those patterns, we attribute features to these graphs using well known graph metrics. These features allow us to describe, in meaningful terms, how different combat tactics contribute to team success. We also present an evaluation of our methodology on the popular MOBA game, DotA 2 (Defense of the Ancients 2). Experiments show that extracted patterns achieve an 80% prediction accuracy when testing on new game logs.

Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning; I.2.1 [Artificial Intelligence]: Applications and Expert Systems—Games

General Terms

Model Development

Keywords

Tactics, Multiplayer Online Battle Arena (MOBA) Game, Sequence of Graphs, Decision Tree

1. INTRODUCTION

Multiplayer online battle arena (MOBA) games are a sub-genre of real-time strategy games in which two teams, typically consisting of five players each, compete against each other with each player controlling a single character. Contrary to real-time strategy games, there is no unit or build-

ing construction in a MOBA game, so much of the strategy revolves around individual character development and cooperative team play in combat. In MOBA games, team-based combat is highly dynamic and difficult to master. That being said, success in combat is often the key to victory in MOBA games. In order to become skilled at combat, players must obtain a vast amount of tactical knowledge as well as invest a large amount of time and effort into practicing [38, 39].

In this paper, we present a data-driven method for identifying patterns in combat which lead to successful game outcomes. We first analyze game logs from professional competitions. We then model combat as a sequence of graphs that each represent how players involved in the combat interacted with each other at different times during the game. These interactions include such actions as dealing damage to enemies and healing or protecting allies. There are several benefits of using a graph representation to describe combat. The most important of these is that the graph representation allows us to use several well established metrics that usually describe graphs to describe combat (*e.g.* vertex in-degree and out-degree [35]). After we extract potential features of combat by computing a set of graph metrics, we use feature selection to identify a set of predictive features and then use them to build a decision tree that predicts the outcome of a game. From this decision tree, we generate a set of rules describing how combat should play out at different times in order for the game to end in victory. Finally, we use these rules to search for frequent subgraphs in the original sequence of graphs in order to identify patterns in combat that are predictive of success.

To characterize the practicality and accuracy of our method, we tested it on the popular MOBA game DotA 2. Game logs of DotA 2 were used both to train our model and to measure its accuracy. An evaluation of the results verified that the identified patterns in combat are predictive of successful game outcomes. Accordingly, our contributions are:

1. Modeling highly-dynamic combat tactics as sequences of graphs, which makes the characteristics of combat easy to capture.
2. Translating the rules generated by a decision tree into combat patterns that are predictive of success, which make combat rules easier to understand.

2. BACKGROUND

In MOBA games, there are generally two types of combat: *farming* and *ganking*. Farming refers to the act of killing weaker, computer-controlled units known as *creeps*. By killing creeps, players gain gold which can then be used to purchase upgrades for their character that will make them stronger. Ganking (gang-killing) is the act of a player or group of players moving around the map and killing enemy players. Like farming, ganking is used as a way to gather gold and experience which will make characters stronger as the game goes on. Ganking has the added effect of hindering the growth of enemy players. This makes ganks extremely important since they can be used to give your team a large advantage at any stage of the game.

As mentioned earlier, MOBA games are typically played with two teams of five players. Each of these players fills a specific *role* on their team. There are many different roles that a player can have, which include:

- **Carry:** Carries are capable of doing a large amount of damage once they have had a chance to develop. This means that they require protection early in the game but are responsible for most of the damage done later in the game.
- **Initiator:** Initiators are characters that usually start large combats by doing large area-of-effect damage or using abilities that can affect the positioning of the enemy team.
- **Disabler:** Disablers have the ability to hinder the functionality of the enemy team by disabling them or controlling them in some way.
- **Tank:** The Tank has the ability to absorb large amounts of damage. This enables other members of the team to live longer in combat.
- **Ganker:** The Ganker has the ability to deal a large amount of damage quickly. Their main goal is to quickly kill enemies so that combat ends as fast as possible.

3. RELATED WORK

Since MOBA games are relatively new, there has been very little analysis done on them. To our knowledge, there has been no attempt to analyze combat in MOBA games up until this point.

Natalia *et al.* [25] used game logs and player communities to analyze the factors of team success in a MOBA game. Tinawat *et al.* [24] analyzed two MOBA games and showed a relation between MOBA games and leadership development. They pointed out that different roles take different types of leadership. The difference between these works and our own is that we are trying to address what specific, in-game behaviors contribute to the success or failure of a team.

While there has been relatively little analysis done on MOBA games, there has been much research done on real-time strategy (RTS) games which are closely related to MOBA games. Typically, these analyses fall into one of two categories: *strategy modeling* or *tactics modeling*. Strategy modeling

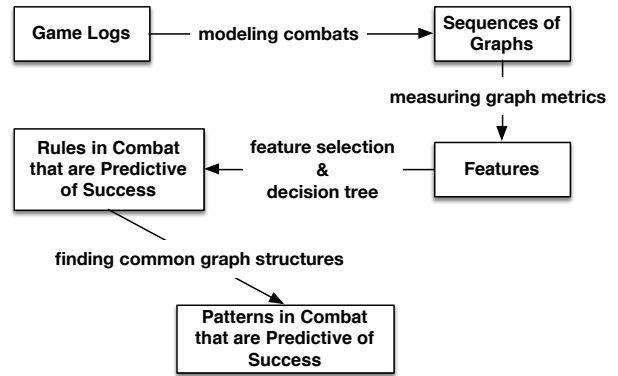


Figure 1: An overview of our methodology. Combat is modeled as a sequence of graphs. Then graph metrics are collected. The features are downselected and used to train a decision tree. Combat rules are then extracted from the tree. Finally, these rules are translated to combat patterns using frequent sub-graph mining.

usually involves trying to achieve the more general goal of determining the best way to spend resources. Questions such as “What buildings should I construct?” and “What units should I produce?” are common with these types of analyses. Examples of these types of analyses include those done by Sailer *et al.* [29], who discovered enemy and friendly strategies by predefining a set of strategies and simulating each step using a Monte Carlo method. In addition to Monte Carlo methods, Hidden Markov Models [10], decision trees [36], and Bayesian Networks [33] have been used to model and predict player strategies in RTS games. Our work is different from these works in that we aim to model behavior at the level of individual combats rather than at the level of team strategy.

More closely related to our work is the work that has been done on tactics modeling. Tactics modeling consists of trying to model how individual units should be used and how combats can be won. There have been several methods used for tactics modeling in the past. These methods include reinforcement learning [37, 26, 31, 19], case-based reasoning [3], Bayesian Networks [32], MDPs [15], and Monte Carlo simulation [7, 4]. While we seek to solve a similar problem, the choice of game genre changes the problem completely. These analyses were performed on RTS games in which one player played against one other player. In this scenario, one does not need to take into account complex team behaviors during combat. This makes the MOBA environment more complex and more difficult to model than that of a traditional RTS game.

4. METHODOLOGY

Our approach for extracting combat patterns that are predictive of success consists of the following steps:

1. Game logs are analyzed and combats are modeled as sequences of graphs.

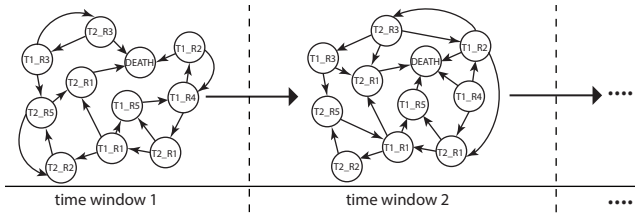


Figure 2: An illustration of combats as a sequence of graphs. Tx_Ry means Role y in Team x . The *death* vertex indicates that a specific role died. Edges indicate that the roles have interacted in some way. Interactions include dealing damage (if between enemies) and healing (if between teammates).

2. Metrics of each sequence of graphs are computed and treated as features that describe combat.
3. We use feature selection to find features of combat that are predictive of winning the game. These features are used to train a decision tree to generate a set of combat rules.
4. These rules are used to find common graph structures that represent combat patterns that are predictive of victory.

These steps are also shown in Figure 1.

4.1 Modeling Combat

Since time is an important aspect of combat in a MOBA game, we have chosen to model combat using a sequence of graphs in which each graph is used to represent at a certain time during the game. The vertices in these graphs correspond to the each of the five roles available on each team (resulting in ten vertices). Edges between vertices indicate that the roles interact in some way. There are two possible interactions possible in this representation: a role on one team doing damage to a role on the other team, or a role on one team healing a role on the same team.

In addition to the ten vertices in the graph corresponding to each role in the game for both teams, we introduce a special vertex that corresponds to death. If a role dies during combat, then an edge is added from that role to the death vertex. This brings the total number of vertices in each graph up to eleven.

In order to model a combat as a sequence of graphs, we need to determine the period of time that each graph represents by defining the size of a time window. There are tradeoffs when choosing a window size. Choosing a small time window generates less complicated individual graphs but more graphs since there are more time windows. On the other hand, a larger time window creates more complicated individual graphs but less of them to deal with. In this work, we use a time window equal to nine minutes. This is because previous work has been done that involved dividing the game into four phases [38]. In our dataset, the average length of a game was 38 minutes. This means that an individual time window would contain roughly nine minutes

each. For the remainder of this paper, we will refer to each of these time windows by the following names: the initial phase, the developing phase, the enhancing phase, the final phase.

Figure 2 shows an example of a sequence of graphs that model a set of combats. In this figure, roles in the graph are referred to by their team number and then their role number. So, $T1_R1$ refers to Role 1 on Team 1. In the first time window, we see directed edges that lead from $T1_R3$ to $T2_R3$ and vice-versa. This indicates that these roles were involved in a combat where they each did damage to each other. Also shown in the figure is an edge from $T2_R3$ to the death node, indicating that this player died. An edge also exists between $T1_R1$ and $T1_R5$, which indicates that $T1_R1$ healed $T1_R5$.

4.2 Computing Graph Metrics as Features

In order to discover patterns inside of combat, we need to first find a set of features that we can use to characterize combat. After modeling combat as a sequence of graphs, we can find features by computing graph metrics on them.

The graph metrics we used are shown below.

In-degree : The number of edges pointing to this vertex [13].

Out-degree : The number of edges that are drawn from this vertex [13].

Closeness : A measure of how long it will take to spread information from a vertex to all other vertices [13].

Betweenness : This quantifies the number of times a vertex acts as a bridge along the shortest path between two other vertices [5, 12].

Eigenvector Centrality : A measure of the influence of a vertex in a graph [23].

Since there are 11 nodes (5 characters * 2 teams + death node), a metric generates 11 features. If there are N parts segmented by a predefined time window in a sequence of graphs and there are a metrics, the total number of features is $11 * a * N$.

4.3 Selecting Features

According to the above graph metrics, if there are 4 parts segmented by a time window in the sequence of graphs, the total number of features is 220 (11 vertices * 5 metrics * 4 windows = 220 features). Not all of these features are predictive of a game win and there is no intuition as to which features would be predictive of game outcomes beforehand. Therefore, we need to perform feature selection before using these features to train a decision tree.

Feature selection [34] identifies important features and removes irrelevant, redundant, or noisy features in order to reduce the dimensionality of the feature space. It can improve efficiency, accuracy and comprehensibility of the models built by learning algorithms.

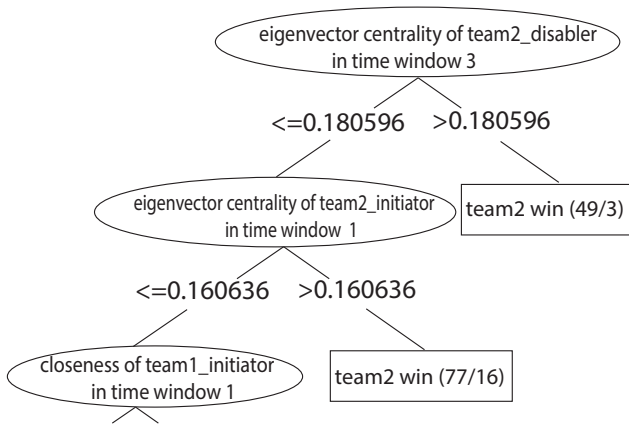


Figure 3: An example decision tree for generating combat rules. Tracing a path from the root node to a leaf node will result in a rule. In each leaf, the first number is the total number of instances reaching the leaf. The second number is the number of those instances that are misclassified.

There are two categories of feature selection methods: *wrapper methods* and *filter methods*. Wrapper methods employ machine learning algorithms to evaluate features, while filter methods use the intrinsic properties of data to assess features [22].

Since we have no prior knowledge as to the quality of any of our features, we chose to use a wrapper feature selection technique based on decision trees. This method begins with an empty set of features and will add features to this set as long as the prediction accuracy of a decision tree built off of them increases. Once the decision tree has achieved a certain accuracy threshold, the features used to train it are returned as the *selected* features.

4.4 Generating Combat Rules

After obtaining a set of predictive combat features, we construct a rule-based classifier using the selected features as inputs. There are many rule-based classification algorithms including IREP [14], RIPPER [9], CN2 [8], RISE [11], AQ [20], ITRULE [30], and decision trees [27]. For the purposes of this work, we have chosen to use decision trees to extract combat rules since it is easy to extract rules by simply tracing a path from the root to the leaves.

We chose to use the C4.5 decision tree algorithm trained on the set of predictive combat features extracted in the previous section. The decision tree is meant to predict the outcome of the game using these combat features. Therefore, the rules generated from this tree are combat rules that will ultimately result in a team winning or losing the game.

Figure 3 contains an example of three levels of a decision tree our algorithm might identify. After we build a decision tree model, tracing a path from the root to the leaves enables us to obtain the rules that are predictive of team success. An example combat rule that can be generated from this tree is that if the eigenvector centrality of the team 2 disabler is greater than 0.180596 then team 2 is likely to win. It is

important to note here that *combat rules* are in terms of the graph metrics that were used to describe combat, which makes them very difficult relate to in-game actions.

The tree generated by the C4.5 algorithm may have many leaves, which means that there are many possible rules that can be generated. While this is true, some of the branches do not represent enough observational instances to be generalizable. To account for this, we use two criteria for identifying usable rules: *confidence* and *support*. Confidence is the percentage of observational instances represented by the node in the decision tree that are associated with a target label. Support is the number of observational instances represented by the node in the tree. The higher the confidence, the more accurate the rule is. The higher the support, the more general the rule is. In this work, we used a confidence threshold of 70% and support threshold of 30. This threshold is based off of experiments done in previous work on generating rules in MOBAs [38, 39].

4.5 Translating to Combat Patterns

After identifying combat rules that are predictive of success, we translate the rules into patterns of specific behavior in combat that are predictive of success by using frequent subgraph mining [18]. Frequent subgraph mining is the process of identifying common substructures in a set of graphs. We refer to the resulting frequent subgraphs as *combat patterns*. The reason we do this is because the behavior patterns contained in these graphs are easier to understand than the combat rules generated earlier. The frequent subgraph mining algorithm we use is detailed below.

Data: Cgs is a corpus of sequences of graphs.

Data: Ru is a combat rule.

Data: W is a set of time windows.

Result: Common graph structures

find subset Ss in Cgs which satisfies Ru ;

foreach w *in* W **do**

 find all condition nodes CN in Ru which are in w ;

 find common graph structures in Ss which involve all condition nodes CN ;

end

Algorithm 1: The algorithm for finding frequent subgraphs that satisfy combat rules.

The condition nodes in a rule consists of the first half of conditions in the rule. For example, consider the rule “IF eigenvector centrality of Disabler on team 2 in time window 3 $>$ 0.180596 THEN team 2 win”. The condition node is “eigenvector centrality of Disabler on team 2 in time window 3”. The a subgraph is considered frequent if the graph structure exists in more than 75% of the set.

Figure 4 illustrates how to identify common graph structures. First, find all condition nodes in a combat rule. Second, find the set of sequences of graphs satisfying the combat rule in the corpus of sequences of graphs. Third, find common graph structures involving the condition nodes in the set. The resulting common graph structures are a combat pattern translated from the combat rule.

5. EXPERIMENTS

Table 1: The summary of role definitions and the formulas used to describe them. The Key Characteristic is the unique characteristic identifying the role in game. Formula is the corresponding method used to identify the role.

Role	Key Characteristic	Formula
Carry	gold-hungry, spends most of time farming in the early game.	$(\text{FarmingTime}/\text{TotalTime})$ in the first half of the game
Ganker	killing as many heroes as possible in the early game.	number of heroes killed in the first half of the game
Initiator	often starting combat.	the number of times the hero deals damage first during combat
Disabler	disabling and controlling other heroes.	number of times the hero casts a detrimental in combat
Tank	sustaining large amounts of damage.	number of items bought that increase defensive characteristics

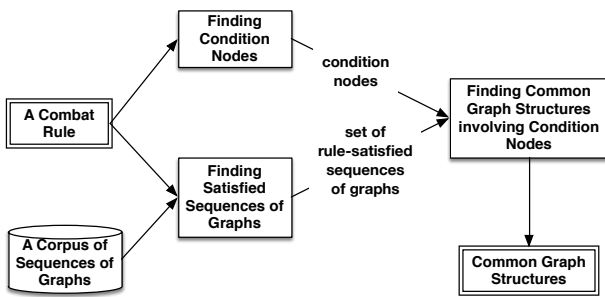


Figure 4: How to identify common graph structures. First, find a set of sequences of graphs satisfying a combat rule in a corpus of sequences of graphs. Second, find common graph structures involving all condition nodes in the set.

We tested our approach on game logs from DotA 2, a popular MOBA game. In the following sections, we will discuss our experimental methodology as well as the results of our study.

5.1 Data Collection

We collected a total of 407 DotA 2 competition game logs from GosuGamers [2] and Getdotastats.com [1], which are online communities for DotA 2 players. The game logs gathered were generated by professional DotA 2 players.

5.2 Experimental Process

To obtain combat information, we used a DotA 2 Replay Parser created by Bruno [6]. The parser takes a game log and outputs a file containing all combat information. While each entry in the file contains several fields, we are only interested in the four fields below:

1. **Replaytime:** The time in that an event occurred.
2. **Attacker:** The unit that deals damage to opponents or heals teammates.
3. **Target:** The unit the receives damage or is healed.

4. **Type:** An integer indicating the type of interaction that occurred (damage, heal, beneficial spell, detrimental spell, and death).

As mentioned earlier, the characters in a MOBA can take on one of five possible roles. Certain characters in these games are better suited to certain roles. In DotA 2, a character can take on many different roles. For example, a character called “Night Stalker” can take on either the Tank or Initiator role. In order to determine which role a character takes in a game log, we examined the types of actions that each character takes and used them to classify each character.

To determine a character’s role, we created a ranking based on the rules in Table 1. The more times a character performs actions that are related to a specific role, the more likely they are to get classified as that role.

Once we have determined what each character’s role was, we are able to use our technique to extract a set of combat rules that are predictive of winning the game. For these experiments, we used the NetworkX library [16] to build the graph model and compute graph metrics. We used the WEKA toolset [17] to perform our feature selection and to build the decision tree model.

5.3 Results

We will discuss the results of each phase of our algorithm in detail below.

5.3.1 Feature Selection

Recall that the first thing that we must do is to select a set of graph metrics that are predictive of winning the game. The set of features resulting from this process is listed below:

- Eigenvector centrality of the Disabler on Team 2 in the enhancing phase
- Eigenvector centrality of the Initiator on Team 2 in the initial phase
- Closeness of the Initiator on Team 1 in the initial phase

Table 2: The summary of identified rules in combat that are predictive of success in DotA 2. *indeg*, *outdeg*, *eige*, and *cls* mean in-degree, out-degree, eigenvector centrality, and closeness respectively. There are four time phases: initial (*ini*), developing (*dev*), enhancing (*enh*), and final (*fin*). *Conf* means confidence. *Win* means the team wins the game. The numeric value in the **IF** statement is the decision boundary generated in decision tree. $m_p_teamX_r$ means m of r in team X in p phase. $m \in \{indeg, outdeg, eige, cls\}$. $p \in \{ini, dev, enh, fin\}$. $X \in \{1, 2\}$. $r \in \{carry, initiator, ganker, disabler, tank\}$.

ID	Conf	Patterns
1	85%	IF $eige_enh_team2_disabler \leq 0.18 \ \& \ eige_int_team2_initiator \leq 0.16 \ \& \ cls_int_team1_initiator \leq 0.32 \ \& \ indeg_dev_team1_ganker \leq 0.1$ THEN team1 win
2	85%	IF $eige_enh_team2_disabler \leq 0.18 \ \& \ eige_int_team2_initiator \leq 0.16 \ \& \ cls_int_team1_initiator \leq 0.32 \ \& \ indeg_dev_team1_ganker > 0.1 \ \& \ cls_dev_team2_tank \leq 0.15$ THEN team2 win
3	91%	IF $eige_enh_team2_disabler \leq 0.18 \ \& \ eige_int_team2_initiator \leq 0.16 \ \& \ cls_int_team1_initiator \leq 0.32 \ \& \ indeg_dev_team1_ganker > 0.1 \ \& \ cls_dev_team2_tank > 0.15 \ \& \ outdeg_dev_team2_carry \leq 0.2$ THEN team1 win
4	97%	IF $eige_enh_team2_disabler \leq 0.18 \ \& \ eige_int_team2_initiator \leq 0.16 \ \& \ cls_int_team1_initiator > 0.32$ THEN team1 win
5	79%	IF $eige_enh_team2_disabler \leq 0.18 \ \& \ eige_int_team2_initiator > 0.16$ THEN team2 win
6	94%	IF $eige_enh_team2_disabler > 0.18$ THEN team2 win

- In-degree of the Ganker on Team 1 in the developing phase
- Closeness of the Tank on Team 2 in the developing phase
- Out-degree of the Carry on Team 2 in the developing phase.

There are many interesting things to note about this set of features. First, betweenness is not a predictive feature. This tells us that indirect relationships among the roles do not have a major effect on a game’s outcome. This means that the important features of combat are all focused on the immediate relationships between roles. Second, there are no features that take place in the final phase of the game. This means that advantages gained earlier in the game are not likely to be overcome by the opposing team late in the game. Finally, all five roles are included in the selected features. This shows that all roles on a team can contribute to the team’s overall success.

5.3.2 Extracting Rules From the Decision Tree

Once we have selected a set of features, we use them as an input into the C4.5 algorithm for creating decision trees. To evaluate the resulting tree, and therefore the rules that are generated from the tree, we performed a 10-fold cross-validation measuring how well the decision tree was able to predict the outcome of the game. Table 3 shows a summary of these results. As you can see in the table, we achieve high accuracy using classification accuracy, sensitivity, specificity, and the area under the ROC curve.

After we have created the tree, we extract combat rules using the method described in Section 4.4. A summary of these rules can be found in Table 2.

5.3.3 Frequent Subgraph Mining

After identifying the set of combat rules that are predictive of success (seen in Table 2), we translate the patterns back

Table 3: Evaluation metrics for 10-fold cross-validation. Classification accuracy (CA), Sensitivity (Sens), Specificity (Spec), and Area under the ROC curve (AUC).

CA	Sens	Spec	AUC
0.8	0.803	0.8	0.83

into specific combat patterns using frequent subgraph mining. Due to paper length, we will only consider the first rule as an example of how these graphs can be interpreted. The frequent graph structures corresponding to rule 1 can be seen in Figure 5.

In Figure 5, the combat behavior for rule 1 is shown. During the initial phase of the game team 1’s Initiator should start the combat before team 2’s Initiator to gain the advantage in combat. In addition to this, the team 1’s Initiator also needs to kill team 2’s Initiator. Additionally, team 1’s Initiator needs to control and disable team 2’s Ganker. In other words, team 1’s Initiator needs to prevent team 2’s Ganker from killing team 1’s Carry since team 1’s Carry is very weak in the early phases of the game. During the developing phase, team 1’s Ganker should dominate the game by attempting to kill every role available on team 2 *except* for the Tank. This makes sense because a team’s Tank is designed to absorb a large amount of damage. This means any attempts to kill the Tank would likely be unsuccessful and would only serve to waste time. During the enhancing phase, the enemy Disabler poses the biggest threat since they can easily inhibit the killing power of the Carry. As a result, team 1’s Ganker and carry should target the Disabler first to ensure that they can continue to do damage throughout combat. During the final phase of the game, there are no patterns. This means that by this point in the game, it is likely too late to perform any actions that will have a considerable impact on the outcome of the game.

6. FUTURE WORK

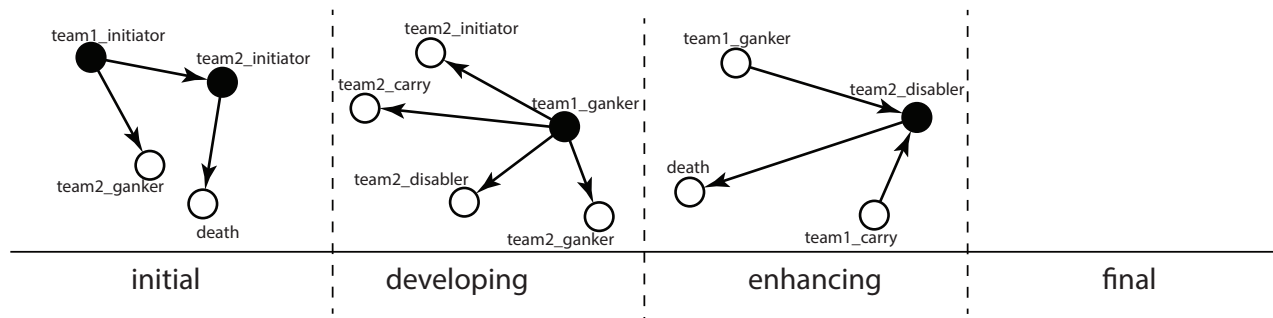


Figure 5: The graphic structures of combat pattern 1. $team_x_y$ means role y in team x . The black nodes are condition nodes explained in Section 4.5.

There are three exciting avenues for future research. First, our method has three free parameters: stopping criterion during feature selection, rule support, and rule confidence. One avenue of future research involves optimizing these parameters using an optimization technique such as genetic algorithms [21] or randomized hill climbing [28]. This way, we can ensure that there is a solid reasoning behind picking a specific threshold value. Second, we would like to further evaluate our combat rules and patterns by using expert players to provide insight into their validity. Third, we hope to use the combat patterns that we discovered to implement a system for assisting players during combats.

7. CONCLUSION

In this paper, we have introduced a data-driven approach for discovering combat patterns of winning teams in MOBA games. We first model combat as a sequence of graphs, and then we then find potential features of combat by computing graph metrics that describe this sequence of graphs. We then select the best features using a wrapper feature selection method. Using these features, we train a decision tree and then extract combat rules from this decision tree. We then use these combat rules to extract frequent subgraphs from the sequences of graphs in order to identify combat patterns that are predictive of winning the game. This technique can be used to gain insight into how players should work together to overcome obstacles in MOBA games and, perhaps, even to teach newer players the skills required to be successful at these types of games.

8. REFERENCES

- [1] Getdotastats.com. <http://getdotastats.com/replays/>.
- [2] GosuGamers. <http://www.gosugamers.net/dota2/replays/>.
- [3] D. W. Aha, M. Molineaux, and M. Ponsen. Learning to win: Case-based plan selection in a real-time strategy game. In *in Proceedings of the Sixth International Conference on Case-Based Reasoning*, pages 5–20. Springer, 2005.
- [4] R.-K. Balla and A. Fern. Uct for tactical assault planning in real-time strategy games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI’09*, pages 40–45, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
- [5] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.
- [6] Bruno. DotA 2 Replay Parser. <http://www.cyborgmatt.com/2013/01/dota-2-replay-parser-bruno/>.
- [7] M. Chung, M. Buro, and J. Schaeffer. Monte carlo planning in rts games. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 117–124, 2005.
- [8] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. *Machine learning—EWSL-91*, 1991.
- [9] W. W. Cohen. Fast Effective Rule Induction. *Machine Learning: Proceedings of the Twelfth International Conference (ML95)*, 1995.
- [10] E. W. Dereszynski, J. Hostetler, A. Fern, T. G. Dietterich, T.-T. Hoang, and M. Udarbe. Learning probabilistic behavior models in real-time strategy games. In V. Bulitko and M. O. Riedl, editors, *AIIDE*. The AAAI Press, 2011.
- [11] P. Domingos. The RISE system: Conquering without separating. *Tools with Artificial Intelligence*, 1994.
- [12] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, Mar. 1977.
- [13] L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, page 215, 1978.
- [14] J. Furnkranz and G. Widmer. Incremental reduced error pruning. *International Conference on Machine Learning*, 1994.
- [15] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational mdps. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI’03*, pages 1003–1010, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.
- [16] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug 2008.
- [17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*,

- 11(1):10–18, Nov. 2009.
- [18] B. Harrison, J. C. Smith, S. G. Ware, H.-W. Chen, W. Chen, , and A. Khatri. Frequent subgraph mining. In N. F. Samatova, W. Hendrix, J. Jenkins, K. Padmanabhan, , and A. Chakraborty, editors, *Practical Graph Mining With R*. Oxford University Press, CRC Press, 2013.
- [19] B. Marthi, S. Russel, and D. Latham. Writing stratagus-playing agents in concurrent alisp. In *Workshop on Reasoning, Representation and Learning in Computer Games, IJCAI-05*, Edinburgh, Scotland, 2005.
- [20] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. *Proc AAAI 1986*, 1986.
- [21] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.
- [22] L. C. Molina, L. Belanche, and À. Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 306–313. IEEE, 2002.
- [23] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, 45:167–256, 2003.
- [24] T. Nuangjumnonga and H. Mitomo. Leadership development through online gaming. 19th ITS Biennial Conference, Bangkok 2012: Moving Forward with Future Technologies - Opening a Platform for All 72527, International Telecommunications Society (ITS), 2012.
- [25] N. Pobiedina, J. Neidhardt, M. d. C. Calatrava Moreno, and H. Werthner. Ranking factors of team success. In *Proceedings of the 22nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 1185–1194, Republic and Canton of Geneva, Switzerland, 2013.
- [26] M. Ponsen and I. P. H. M. Spronck. Improving adaptive game ai with evolutionary learning. In *University of Wolverhampton*, pages 389–396, 2004.
- [27] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.
- [28] S. Russell and P. Norvig. *Artificial intelligence: a modern approach (2nd edition)*. Prentice Hall, 2003.
- [29] F. Sailer, M. Buro, and M. Lanctot. Adversarial planning through strategy simulation. In *IEEE Symposium on Computational Intelligence and Games, 2007. CIG 2007*, pages 80–87, 2007.
- [30] P. Smyth and R. M. Goodman. An information theoretic approach to rule induction from databases. *Knowledge and Data Engineering, IEEE Transactions on*, 4(4):301–316, 1992.
- [31] P. Spronck, I. G. Sprinkhuizen-Kuyper, and E. O. Postma. On-line adaptation of game opponent ai with dynamic scripting. *Int. J. Intell. Games and Simulation*, 3(1):45–53, 2004.
- [32] M. Stanescu, S. P. Hernandez, G. Erickson, R. Greiner, and M. Buro. Predicting army combat outcomes in starcraft. In *Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013.
- [33] G. Synnaeve and P. Bessiere. A bayesian model for opening prediction in rts games with application to starcraft. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 281–288, 2011.
- [34] F. Tan. Improving feature selection techniques for machine learning. 2007.
- [35] S. G. Ware. An introduction to graph theory. In N. F. Samatova, W. Hendrix, J. Jenkins, K. Padmanabhan, , and A. Chakraborty, editors, *Practical Graph Mining With R*. Oxford University Press, CRC Press, 2013.
- [36] B. G. Weber and M. Mateas. A data mining approach to strategy prediction. In *IEEE Symposium on Computational Intelligence and Games, 2009. CIG 2009*, pages 140–147. IEEE, 2009.
- [37] A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Learning and transferring roles in multi-agent reinforcement. In *Proc. AAAI-08 Workshop on Transfer Learning for Complex Tasks*, 2008.
- [38] P. Yang and D. L. Roberts. Extracting human-readable knowledge rules in complex time-evolving environments. In *Proceedings of The 2013 International Conference on Information and Knowledge Engineering (IKE 13)*, Las Vegas, Nevada USA, July 2013.
- [39] P. Yang and D. L. Roberts. Knowledge discovery for characterizing team success or failure in (a)rts games. In *Proceedings of the IEEE 2013 Conference on Computational Intelligence in Games (IEEE CIG)*, Niagara Falls, Canada, August 2013.