Case-Based Goal Formulation

Ben G. Weber and **Michael Mateas** and **Arnav Jhala**
Expressive Intelligence Studio
University of California, Santa Cruz
{bweber, michaelm, jhala}@soe.ucsc.edu

Abstract

Robust AI systems need to be able to reason about their goals and formulate new goals based on the given situation. *Case-based goal formulation* is a technique for formulating new goals for an agent using a library of examples. We provide a formalization of this term and two algorithms that implement this definition. The algorithms are compared against instance-based and model-based techniques on the tasks of opponent modeling and strategy selection in the real-time strategy game *StarCraft*. Our system, *EISBot*, implements these techniques and is capable of consistently defeating the built-in AI of *StarCraft*.

Introduction

One of the requirements for creating robust real-world AI applications is building systems capable of deciding which actions should be performed to pursue a goal. Goal formulation is a technique for an agent to determine which goals need to be achieved. The major challenges in goal formulation are developing representations for the agent to reason about, recognizing when new goals need to be formulated due to plan failure, and operating in a real-time environment.

Contemporary computer games are an excellent domain for research in this area, because they offer rich, complex domains for AI researchers (Laird and VanLent 2001). Games resemble the real world in that they are real-time, contain huge decision spaces, and enforce imperfect information. Real-time strategy (RTS) games in particular present interesting research challenges (Buro 2003), such as reasoning about both strategic and tactical goals simultaneously. Performing well in RTS games requires long-term planning. However, an agent's goals can become invalidated due to player interaction.

One of the benefits of using real-time strategy games is the amount of gameplay data available for analysis. Thousands of professional-level replays are available for games such as *StarCraft* (Weber and Mateas 2009a). Developing techniques for automatically extracting domain knowledge from game replays is expected to help automate the process of building game AI (Ontanón et al. 2010) as well as lead to more interesting computer opponents that learn a variety of gameplay styles. The major challenge in harnessing this data is dealing with the limited amount of information available: traces contain raw game state and do not contain a player's goals or intentions.

In this paper we introduce the term *case-based goal formulation*. This term refers to performing goal formulation based on retrieval and adaptation of cases from a library of examples. Case-based goal formulation is inspired by techniques in the case-based reasoning (Aamodt and Plaza 1994) and machine learning literature. The goal of this technique is to automate the process of performing goal formulation by harnessing a corpus of data. We provide two knowledge-weak implementations of case-based goal formulation. The Trace algorithm performs goal formulation by retrieving the most relevant case and building a new goal state based on the actions performed in the retrieved case. The MultiTrace algorithm is an extension of the Trace algorithm that retrieves multiple traces and combines the results.

Related Work

Goal formulation has been applied to building game AI. The RTS game *Master of Orion 3* used a goal-based architecture to make high-level strategic decisions (Dill and Papp 2005). The agent's goal formulation is referred to as a "think" process that executes once every 30 seconds. Goal formulation can also be triggered by important game events, such as capturing an enemy city. The system applies *goal inertia* and *goal commitment* techniques to prevent the agent from dithering between strategies.

Goal-oriented action planning (GOAP) has been applied to first person shooter games (Orkin 2003). In a GOAP architecture, each non-player character has a set of goals that can be activated based on relevance. When a goal is triggered by its activation criteria, the system builds a plan to achieve it. The main challenges in applying GOAP to game AI are developing suitable world representations and planning operators that support near real-time operation. Additionally, GOAP architectures tend to create short-term plans.

Case-based planning is a another technique that can be applied to building goal-based game AI. *Darmok* is a case-based planner that uses game traces to interleave planning and execution in the RTS game *Wargus* (Ontanón et al. 2010). Cases are extracted from human-annotated traces and

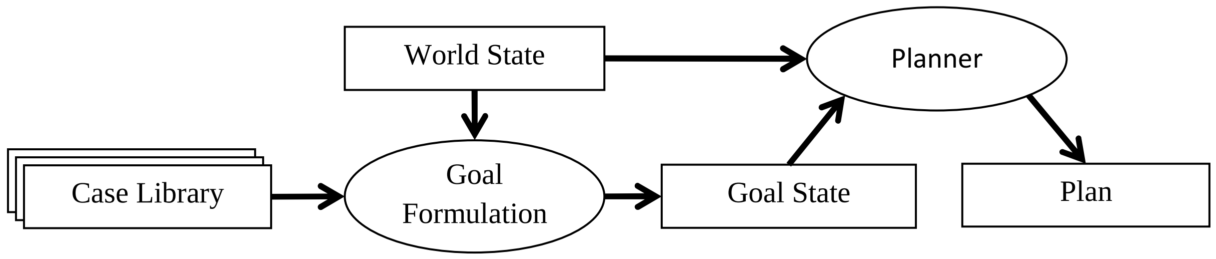

Figure 1: Case-based goal formulation makes use of a case library to formulate a goal state. To achieve this goal state the system computes the actions required to reach the goal state from the current state and then builds a totally-ordered plan.

specify primitive actions and subgoals required to achieve a goal. The system initially has a single goal of winning the game, which it achieves by retrieving and adapting cases from the library to build hierarchical plans. *Darmok* differs from our approach in that our case representation contains a goal state, while *Darmok* cases contain the actions needed to a achieve a specific goal. Additionally, our approach does not require defining a goal ontology or annotating traces.

Case-Based Goal Formulation

Case-based goal formulation is a technique for performing goal formulation based on a collection of cases. It is motivated by the goal of reducing the amount of domain engineering required to build autonomous agents. For example, the *EISBot* contains no pre-authored knowledge of strategic reasoning in StarCraft, but learns this knowledge automatically from case-based goal formulation.

An overview of case-based goal formulation is shown in Figure 1. The inputs to the system are the current world state and the case library. The task of the goal formulation component is to determine a new goal state for the agent to pursue. The world state and goal state are then passed to the planner, which determines the actions necessary to reach the goal state from the current world state. The output of the system is a totally-ordered plan for the agent to execute.

We refer to the number of actions in the generated plan as the planning window size. The motivation for retrieving a set of actions versus a single action is to enable a tradeoff between plan size and re-planning. A small planning window should be used in domains where plans are invalidated frequently, while a large planning window should be used in domains that require long-term plans.

Case-based goal formulation resembles classification and case-based planning. In the case that the planning window size is set to 1, our technique is similar to classification algorithms. This results from goal formulation retrieving a single action to execute, which eliminates the need for planning. In case-based planning (Cox, Muñoz-Avila, and Bergmann 2006), the agent's goal is defined before retrieval and the retrieval process consists of building a plan to achieve the agent's goal. In case-based goal formulation, the planning process is decoupled from the case-retrieval process.

Formalization

We define goal formulation as follows:

> given the world state, s, and the agent's current goal state, g, formulate the agent's new goal state, $g\prime$, after executing n actions in the world

where n is the planning window size[1]. Case-based goal formulation is a technique for implementing goal formulation. It is defined as follows:

> the agent's new goal state, $g\prime$, is computed by retrieving the most similar case, q, to the current goal state, g, and adding the difference between q and its future state, $q\prime$, which is the state after n actions have been applied to the case.

Formally:

$$q = min(distance(g, c))$$
$$g\prime = g + (q\prime - q)$$

where c is a case in the case library and the distance function may be a domain independent or domain specific distance metric.

Trace Algorithm

The Trace algorithm is a technique we developed for implementing case-based goal formulation using traces of world state. A trace is a list of tuples containing world state and actions, and is a single episode demonstrating how to perform a task in a domain. For example, a game replay is a trace that contains the current game state and player actions executed each game frame, which demonstrates how to perform a specific task in the game.

The algorithm utilizes a case representation where each case is an unlabeled feature vector which describes the world state at a specific time. The algorithm is capable of determining the actions performed between different time steps by analyzing the difference between feature vectors. Note that computing the actions performed between time steps is trivial in our example domain, because each action in this domain corresponds to incrementing or decrementing a single feature. However, this task may be non-trivial in domains with actions that modify multiple features and becomes a planning problem.

Cases from a trace are indexed using the time step feature. This enables efficient lookup of $q\prime$ once a case, q, has been

[1]Goal formulation has been more generally defined as creating a goal, in response to a set of discrepancies, given their explanation and the current state (Muñoz-Avila et al. 2010).

selected. Assuming that the retrieved case occurred at time t in the trace, $q\prime$ is defined by the world state at time $t+n$. Since the algorithm uses a feature vector representation, $g\prime$ can be computed as follows:

$$\begin{aligned} q &= q_t \\ q\prime &= q_{t+n} \\ g\prime(x) &= g(x) + (q\prime(x) - q(x)) \end{aligned}$$

where x is a feature in the case representation.

To summarize, the Trace algorithm works by retrieving the most similar case, finding the future state in the trace based on the planning window size, and adding the difference between the retrieved states to the current goal state.

Example

Consider an agent with a planning window of size 2, a Euclidean distance function, and the following goal state:

$$g =< 3, 0, 1, 1 >$$

There is a single trace, consisting of the following cases:

$$\begin{aligned} q_1 &=< 2, 0, 0.5, 1 > \\ q_2 &=< 3, 0, 0.7, 1 > \\ q_3 &=< 4, 1, 0.9, 1 > \\ q_4 &=< 4, 1, 1.1, 2 > \end{aligned}$$

The Trace algorithm would proceed as follows:

1. The system retrieves the most similar case: q_2
2. $q\prime$ is retrieved: $q\prime = q_{2+n} = q_4$
3. The difference is computed: $q\prime - q =< 1, 1, 0.4, 1 >$
4. $g\prime$ is computed: $g\prime = g + (q\prime - q) =< 4, 1, 1.4, 2 >$

After goal formulation, the agent's goal state is set to $g\prime$.

MultiTrace Algorithm

The MultiTrace algorithm is an extension of the Trace algorithm in which multiple cases are retrieved when formulating a goal state. The technique is similar to k-NN, where the k most similar cases are retrieved. The intention of combining multiple traces for goal formulation is to deal with new situations that may not be present in the case library. The algorithm is defined as follows:

$$w_j = e^{-\alpha * distance(g, q_j)}$$

$$\sum_{j=1}^{k} w_j = 1$$

$$g\prime(x) = g(x) + \sum_{j=1}^{k} w_j * (q_j\prime(x) - q_j(x))$$

where α is a parameter for tuning case relevance[2]. Each of the k retrieved cases is assigned a weight based on the distance to the current goal state. The weights are then normalized. The cases are combined into a single goal state by multiplying each retrieved case by its weight.

[2]Functions other than exponential weighting can be used.

Application to RTS Games

We applied case-based goal formulation to the RTS game *StarCraft*[3]. This game was selected, because it provides a complex domain with a large strategy space and there are a huge number of professional replays available for building a case library. Case-based goal formulation was used for performing opponent modeling and strategy selection.

Case Representation

Our case representation is a feature vector that tracks the number of units and buildings that a specific player controls. There is a feature for each unit and building type and the value of each feature is the number of that type that have been produced since the start of the game. Since there is an adversarial player in *StarCraft*, the goal state encodes only a single player's state. The system encodes the agent's state for strategy selection and the opponent's state for opponent modeling.

Table 1: An example trace showing when a player performed build and train actions.

Frame	Player	Action
100	1	Train SCV
300	1	Build Supply Depot
500	1	Train SCV
700	1	Build Barracks
900	1	Train Marine

We collected thousands of professional-level replays from community websites and converted them to our case representation. Replays were converted from Blizzard's proprietary binary format into text logs of game actions using a third-party tool. A subset of an example trace is shown in Table 1. An initial case, q_1, is generated with all values set to zero, except for the worker unit type (SCV) and command center type, which are set to 4 and 1 respectively, because the player begins with these units. A new case is generated for each action that trains a unit or produces a building. The value of the new case is initially set to the value of the previous case, then the feature corresponding to the train or build action is incremented by one. Considering a subset of the features (# SCVs, # Supply Depots, # Barracks, # Marines), the example trace would produce the following cases:

$$\begin{aligned} q_1 &=< 4, 0, 0, 0 > \\ q_2 &=< 5, 0, 0, 0 > \\ q_3 &=< 5, 1, 0, 0 > \\ q_4 &=< 6, 1, 0, 0 > \\ q_5 &=< 6, 1, 1, 0 > \\ q_6 &=< 6, 1, 1, 1 > \end{aligned}$$

Our case library consists of 1,831 traces and 244,459 cases.

[3]StarCraft and its expansion StarCraft:Brood War were developed by Blizzard Entertainment^{TM}

Evaluation

We evaluated our approach by applying it to opponent modeling in *StarCraft*. Opponent modeling was performed by executing goal formulation on the opponent's state. Given the opponent's current state, g, an opponent modeling algorithm builds a prediction of the opponent's future state, $p\prime$, by applying n actions to g. This prediction is then compared against the opponent's actual state n actions later in the game trace, $g\prime$. All experiments computed error using the root mean squared error (RMSE) between the predicted goal state, $p\prime$, and the opponent's actual goal state, $g\prime$.

Experiments used 10-fold cross validation. A modified version of fold-slicing was utilized to prevent cross-fold trace contamination, where cases from the same trace are present in both training and testing datasets. To get around this problem, all cases from a trace are always included in the same fold. We had sufficient training data for the folds to remain relatively balanced.

Case-based goal formulation was compared against classification algorithms. The classification case representation contains an action in addition to the goal state, which serves as a label for the case. The following algorithm was applied to build predictions with a planning window of size n:

```
p′ = goal(state g, int n)
        if (n == 0) return g
        else return goal(g + c(g), n-1)
```

where goal is the formulation function, c(g) refers to classifying an instance, and g + c(g) refers to updating the goal state by applying the action contained in the case. The goal function runs the classifier, updates the state based on the prediction, and repeats until n classifications have been performed.

We evaluated the following algorithms: *Null* predicts $p\prime = g$ and serves as a baseline, *IB1* uses a nearest neighbor classifier (Aha, Kibler, and Albert 1991), *AdaBoost* uses a boosting classifier (Freund and Schapire 1996), *Trace* uses our Trace algorithm with a Euclidean distance metric, and *MultiTrace* uses our MultiTrace algorithm with a Euclidean distance metric. Weka implementations were used for the IB1 and AdaBoost classifiers (Witten and Frank 2005).

The first experiment evaluated opponent modeling on various planning window sizes at different stages in the game. The different stages in the game refer to how many train and build actions have been executed by the player so far. Different stages in the game were simulated by building predictions for the cases indexed at a specific time from the traces in the test dataset. Opponent modeling was applied to predicting a Terran player's actions in Terran versus Protoss matches[4].

Results from the first experiment are shown in Figure 2. The results show that the Trace and MultiTrace algorithms outperformed the classification algorithms in all of the experiments. The Trace and MultiTrace algorithms perform similarly, except in the range of 10 to 30 game actions. In fact, all of the algorithms performed poorly in this range except the MultiTrace algorithm. Our hypothesis is that it is difficult to perform opponent modeling at this stage of the game, because it is the time at which players begin to work towards a specific strategy.

The second experiment evaluated the effects of adding additional features to the case representation. The additional features specify the game frame in which the player first produces a specific unit type or building type (Weber and Mateas 2009a). There is a timing feature for each of the original features. The different feature sets include the original feature set, the addition of the player timing features (timing), the addition of the opponent timing features (opponent timing), and the addition of both player and opponent timing features (both timing). Results from the second experiment are shown in Figure 3. The results show that adding any of the additional feature sets greatly improves opponent modeling in the range of 10 to 30 game actions and that adding timing information caused the Trace algorithm to perform slightly better in this range.

Implementation

We implemented case-based goal formulation in a *StarCraft* playing agent, *EISBot*. The agent consists of two components: a goal formulation component that performs strategy selection, and a reactive planner that handles second-to-second actions in the game. *EISBot* interfaces with *StarCraft* using the Brood War API. Currently, *EISBot* plays only the Protoss faction.

The goal formulation component uses the Trace algorithm with the player timing feature set. The agent uses an initial planning window of size 40, and reduces the window size to 20 in subsequent formulations. A larger window is used initially, because the plan to achieve the agent's initial goal is unlikely to be invalidated by the opponent in this stage of the game. The later window size of 20 is used to prevent the agent from dithering between strategies. Goal formulation is triggered by the following events: the current plan completes execution, the agent or the opponent builds an expansion, or the agent or the opponent initiates an attack. After goal formulation, the agent's current plan is overwritten with the newly formulated plan. Generated plans contain the train and build actions for the agent to perform.

Our current implementation of EISBot does not use a planner. Since EISBot retrieves single traces, it sequences the actions based on the order in which they were performed in the trace. This is still a form of goal formulation, where the agent retrieves both a goal and a plan to achieve the goal.

The reactive portion of *EISBot* is written in the reactive planning language ABL (Mateas and Stern 2002). The agent's behavior is composed of several managers that handle different aspects of gameplay (McCoy and Mateas 2008). For example, the tactics manager handles combat, while the worker manager handles resource gathering. *EISBot* interfaces with the goal formulation component through working memory, which serves as a blackboard. Our approach is similar to previous work, which interfaces ABL with a case-based reasoning component (Weber and Mateas 2009b). McCoy and Mateas's integrated agent design was initially applied to *Wargus*, but transferred well to *Star-*

[4]*StarCraft* contains three factions: Protoss, Terran, and Zerg

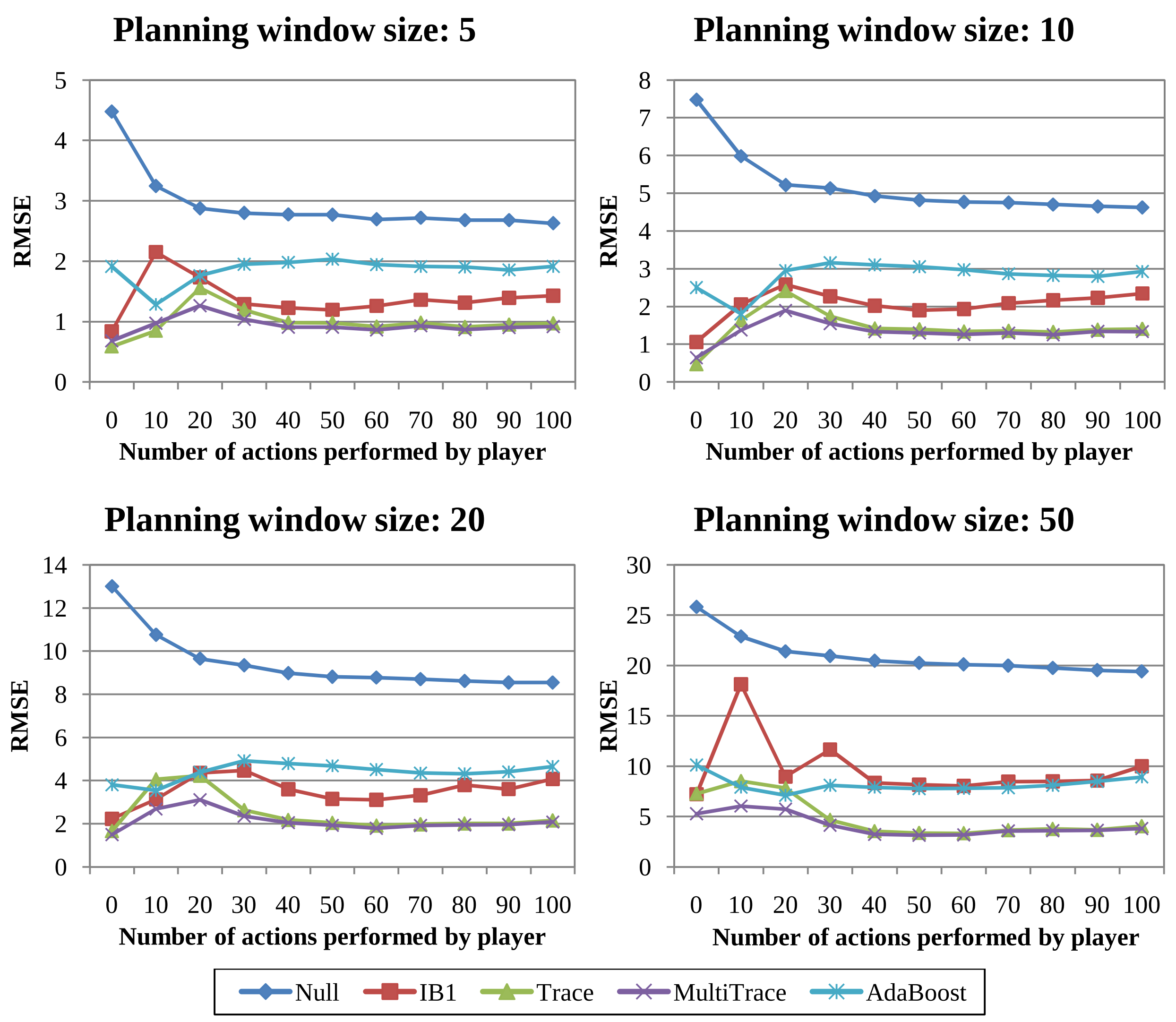

Figure 2: Root mean-squared error (RMSE) of the algorithms on various planning window sizes. The horizontal axis refers to the number of train and build actions that have been executed by the player.

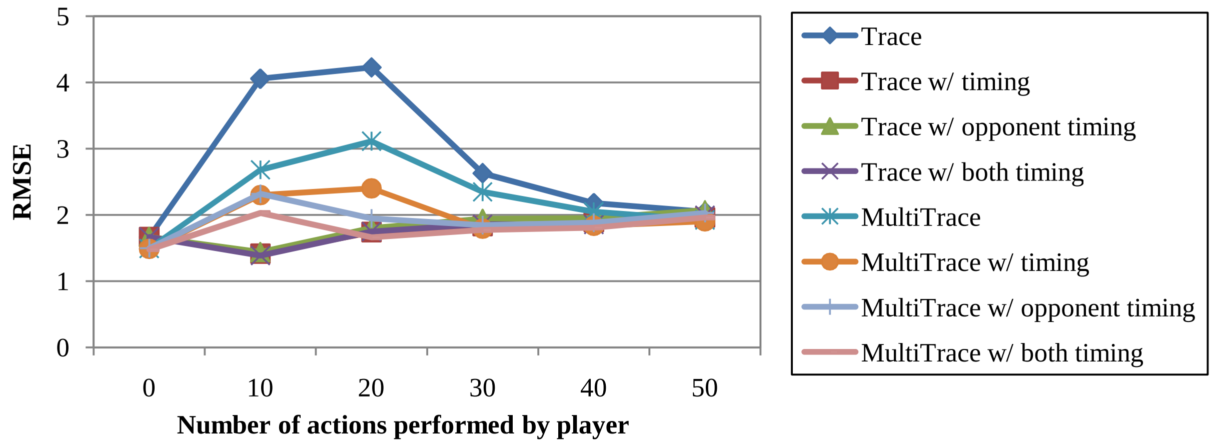

Figure 3: Error rates of the Trace and MultiTrace algorithms for a window size of 20. Each algorithm was evaluated with four different feature sets that include the original features and additional timing features.

Table 2: Results versus the built-in *StarCraft* AI

	Versus			
	Protoss	Terran	Zerg	Overall
Win-loss record	7-13	20-0	4-16	31-29
Win ratio	0.35	1.00	0.20	0.52

Table 3: Results versus human players

	Versus			
	Protoss	Terran	Zerg	Overall
Win-loss record	3-20	2-15	2-8	7-43
Win ratio	0.13	0.12	0.20	0.14

Craft. The main change required was the addition of micro-management behaviors in the tactics manager.

We evaluated *EISBot* versus the built-in AI of *StarCraft* as well as human players on a *StarCraft* ladder server. All matches were played on the map Python, which has been used in professional gaming tournaments. Results versus the built-in AI are shown in Table 2. Our agent was able to consistently defeat Terran opponents, but had less success versus the other factions. *EISBot* lost to Protoss and Zerg opponents due to lack of sufficient behaviors for handling unit formations and grouping. Results versus human players are shown in Table 3. While *EISBot* won only 14% of matches, it is important to note that the agent was evaluated on a highly competitive ladder server. Also, players were notified that they were playing a bot, which may have caused players to harass it for an easy victory.

Conclusions and Future Work

Case-based goal formulation is a technique for creating goals for an agent to achieve, which resembles case-based reasoning and instance-based techniques. The process formulates goal states based on a library of examples. This technique is useful for domains where there is an abundance of data and domain engineering is challenging.

We presented two algorithms for implementing case-based goal formulation. The algorithms were shown to outperform classification techniques in opponent modeling. We also presented an implementation of our technique in a complete game playing agent, *EISBot*, that consistently defeats the built-in AI of *StarCraft* and occasionally defeats competitive human players.

While we applied case-based goal formulation to the domain of real-time strategy games, the technique could be generalized to other domains as well. Case-based goal formulation provides an implementation of the goal formulation component in the goal driven autonomy conceptual model (Muñoz-Avila et al. 2010).

There are two main research directions for future work in this area. The first direction is to investigate the application of a conventional planner to our agent. One of the benefits to using a planner would be the application of additional domain knowledge, such as adding the unit dependencies necessary to achieve a goal state or factoring in state from the reactive planner. The second direction is to evaluate the potential of our approach in transfer learning tasks, such as playing all three factions in *StarCraft*.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications* 7(1):39–59.

Aha, D.; Kibler, D.; and Albert, M. 1991. Instance-based learning algorithms. *Machine Learning* 6(1):37–66.

Buro, M. 2003. Real-Time Strategy Games: A New AI Research Challenge. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1534–1535.

Cox, M.; Muñoz-Avila, H.; and Bergmann, R. 2006. Case-based planning. *The Knowledge Engineering Review* 20(03):283–287.

Dill, K., and Papp, D. 2005. A Goal-Based Architecture for Opposing Player AI. In *Proceedings of the Artificial Intelligence for Interactive Digital Entertainment Conference*. AAAI Press.

Freund, Y., and Schapire, R. E. 1996. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, 148–156. San Francisco: Morgan Kaufmann.

Laird, J., and VanLent, M. 2001. Human-level AI's killer application: Interactive computer games. *AI magazine* 22(2):15.

Mateas, M., and Stern, A. 2002. A Behavior Language for Story-Based Believable Agents. *IEEE Intelligent Systems* 17(4):39–47.

McCoy, J., and Mateas, M. 2008. An integrated agent for playing real-time strategy games. In *Proceedings of the 23rd national conference on Artificial intelligence (AAAI)*, 1313–1318. AAAI Press.

Muñoz-Avila, H.; Aha, D.; Jaidee, U.; Klenk, M.; and Molineaux, M. 2010. Applying goal directed autonomy to a team shooter game. In *Proceedings of the Florida Artificial Intelligence Research Society Conference*. AAAI Press.

Ontanón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2010. On-Line Case-Based Planning. *Computational Intelligence* 26(1):84–119.

Orkin, J. 2003. *AI Game Programming Wisdom 2*. Charles River Media, S. Rabin editor. chapter Applying Goal-Oriented Action Planning to Games, 217–228.

Weber, B., and Mateas, M. 2009a. A Data Mining Approach to Strategy Prediction. In *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games*, 140–147. IEEE Press.

Weber, B., and Mateas, M. 2009b. Case-Based Reasoning for Build Order in Real-Time Strategy Games. In *Proceedings of the Artificial Intelligence for Interactive Digital Entertainment Conference*, 106–111. AAAI Press.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*. San Francisco, California: Morgan Kaufmann.