Defeating Novel Opponents in a Real-Time Strategy Game

Matthew Molineaux, David W. Aha, and Marc Ponsen
ITT Industries; AES Division; Alexandria, VA 22303
Navy Center for Applied Research in Artificial Intelligence;
Naval Research Laboratory (Code 5515); Washington, DC 20375
Department of Computer Science and Engineering;
Lehigh University; Bethlehem, PA 18015
1first.last@nrl.navy.mil 3mjp304@lehigh.edu

Abstract
The Case-Based Tactician (CAT) system, created by Aha, Molineaux, and Ponsen (2005), uses case-based reasoning to learn to win the real-time strategy game Wargus. Previous work has shown CAT’s ability to defeat a randomly selected opponent from a set against which it has trained. We now focus on the task of defeating a selected opponent while training on others. We describe CAT’s algorithm and report its cross-validation performance against a set of Wargus opponents.

1 Introduction
Research on artificial intelligence (AI) and games has an extraordinary history that dates from 1950. Automated game-playing programs now exist that outperform world champions in classic board games such as checkers, Othello, and Scrabble (Schaeffer, 2001). These efforts brought about significant advancements in search algorithms, machine learning techniques, and computer hardware.

In recent years, AI researchers (e.g., Laird & van Lent, 2001; Buro, 2003) have begun focusing on complex strategy simulation games that offer new challenges, including but not limited to partially observable environments, asynchronous gameplay, comparatively large decision spaces with varying abstraction levels, and the need for resource management processes.

Learning to win such games may require more sophisticated representations and reasoning capabilities than games with smaller decision spaces. To assist their studies, Ponsen and Spronck (2004) developed a lattice for representing and relating abstract states in Wargus, a moderately complex real-time strategy game which mimics the popular commercial game Warcraft II. They also sharply reduced the decision space by employing a high-level language for game agent actions. Together, these constrain the search space of useful plans (i.e., strategies) and state-specific subplans (i.e., tactics). This approach allowed them to focus on the game at a strategic level, abstracting away the minutiae of individual actions. They developed a genetic algorithm and a technique called dynamic scripting to learn plans spanning the entire game, which win against a fixed opponent. We build on this success and approach the same challenge, without foreknowledge of the adversary.

We have developed a case-based approach for playing Wargus that learns to select appropriate tactics based on the current state of the game. This approach is implemented in the Case-based Tactician (CAT) system. We will report learning curves that demonstrate that its performance improves against one opponent from training against others. CAT is the first case-based system designed to defeat an opponent that uses tactics and strategies that it has not trained against.

In Section 2, we review some case-based approaches in games research. In Section 3, we describe the domain knowledge available to CAT and how this reduces the complexity of finding effective strategies for defeating Wargus opponents. CAT’s case-based approach is examined in section 4. We review our empirical methodology and CAT’s results in Section 5, and close with a discussion in Section 6 that highlights several future research objectives.

2 Background: Case-based Reasoning Research in Large Decision-Space Games
Below, we contrast some of the prior Case-Based Reasoning research focusing on large decision-space games with CAT.

ROBOCUP SOCCER is a popular CBR focus. Wendler and Lenz (1998) described an approach for identifying where simulated agents should move, while Wendler et al. (2001) reported strategies for learning to pass. Gabel and Veloso (2001) instead used a CBR approach to select members for a team. Karol et al. (2003) proposed a case-based action selection algorithm for the 4-legged league. While these real-time environments are challenging strategically, they do not involve complicating dimensions common to strategy games, such as economies, research, and warfare.

Some researchers have addressed real-time individual games. Goodman (1994) applied a projective visualization approach for Bilestoad, a personal combat game, to predict actions that would inflict damage on the adversary and/or minimize damage to oneself. Fagan and Cunningham (2003) instead focused on a plan recognition task; they acquire cases (state-action planning sequences) for predicting the next action of a human playing Space Invaders™. In
contrast, CatT does not perform projective modeling, and does not learn to recognize adversarial plans. Instead, it acquires cases concerning the application of a subplan in a given state, learns to select subplans for a given state, and executes them in a more complex gaming environment.

Fasciano’s (1996) MAYOR learns from planning failures in SimCity™, a real-time city management game with no traditional adversaries. MAYOR monitors planning expectations and employs a causal model to learn how to prevent failure repetitions, where the goal is to improve the ratio of successful plan executions. In contrast, CAT does not employ plan monitoring or causal goal models, and does not adapt retrieved plans. Rather, it simply selects, at each state, a good tactic (i.e., subplan) to retrieve. Also, our gaming environment includes explicit adversaries.

Ulam et al. (2004) described a meta-cognitive approach that performs failure-driven plan adaptation for Freeciv, a complex turn-based strategy game. While they employed substantial domain knowledge in the form of task models, it was only enough to address a simple sub-task (defending a city). In contrast, CAT performs no adaptation during reuse, but does perform case acquisition. Also, CAT focuses on winning a game rather than on performing a subtask.

Guestrin et al. (2003) applied relational Markov decision process models for some limited Wargus scenarios (e.g., 3x3 combat). They did not address more complex scenarios because their planner’s complexity grows exponentially with the number of units. Similarly, Cheng and Thawonmas (2004) proposed a case-based plan recognition approach for assisting Wargus players, but only for low-level management tasks. Their state representation is comprehensive and incorporates multiple abstraction levels.

3 Domain Knowledge in CatT

CAT employs three sources of domain knowledge to reduce the complexity of WARGUS. Two of these are from Ponsen and Sprock (2004): a building state lattice, which abstracts the state space, and a set of tactics for each state.

In Wargus, certain buildings provide the player with new capabilities, in the form of newly available technologies, units and buildings. Therefore, it makes sense to break the game up into periods when a certain set of buildings exist. We call the time between the construction of one such building to the time the next is built a building state. The building state defines the set of actions available to the player at any one time. It is important not to confuse this with the game state, which encompasses all of the variables of the game, and changes much more frequently, whether the player takes action or not.

The building state lattice (Figure 1) shows the transitions that are possible from one building state to the next. This was developed in the course of Ponsen and Spronck’s research for the purpose of planning. In their research, plans spanned an entire game, which we call strategies. These strategies are made up of tactics, which are subplans that span a single building state. The tactics are made up of individual actions; using the state lattice, Ponsen and Sprock ensured that all actions used were legal for the corresponding building state, and that the entire plan was therefore legal.

The second source of domain knowledge CAT has access to (i.e., a set of tactics for each state) was acquired using Ponsen and Sprock’s genetic algorithm (2004). This was used to evolve chromosomes, representing counter-strategies, against specific opponents. Eight opponent strategies (see Table 1) were available, including some that are publicly available and others that were manually developed. The resulting counter-strategies (i.e., one for each of the eight opponents) were used as a source for automatically acquiring the tactics (Ponsen et al. 2005a) used by CAT. The names of counter-strategies are shown in the lower left of Figure 1. By making reference to the state lattice, CAT is able to determine what tactics are applicable whenever a new building state is entered and choose among them. Figure 1 shows for each building state which of the

Figure 1 shows the transitions that are possible from one building state to the next. This was developed in the course of Ponsen and Spronck’s research for the purpose of planning. In their research, plans spanned an entire game, which we call strategies. These strategies are made up of tactics, which are subplans that span a single building state. The tactics are made up of individual actions; using the state lattice, Ponsen and Spronck ensured that all actions used were legal for the corresponding building state, and that the entire plan was therefore legal.
tactics are applicable to each state. For example, State 1 offers tactics from all evolved counter-strategies, only three tactics apply to state 20, namely those derived from the evolved_SC4, evolved_LBLA and evolved_SR counter-strategies.

In order to choose which tactic to use, CAT employs a casebase for each building state which records certain features of the game state, a value indicating which tactic was selected, and an associated performance value (see section 4.3) achieved by using that tactic in gameplay. We will discuss our case-based strategy selection algorithm in more detail in the next Section.

4 Case-Based Strategy Selection

Our case-based approach for selecting which tactic to use in each state employs the state lattice and state-specific tactics libraries described in Section 3. By doing this, the decision space (i.e., the number of tactics per state) becomes small, and an attribute-value representation of game situations suffices to select tactics. We define a case $C$ as a tuple of four objects:

$C = <$Building State, Description, Tactic, Performance$>

where Building State is an integer node index in the building state lattice, Description is a set of features of the current situation (see Table 2), Tactic is a counter-strategy’s sequence of actions for that building state, and Performance is a real value in the range [0,1] that reflects the utility of choosing that tactic for that building state, where higher values indicate higher performance (see Section 4.3). We next use Aamodt and Plaza’s (1994) task decomposition model to detail our approach.

4.1 Retrieval

CAT retrieves cases when a new state in the lattice is entered (i.e., at the game’s start, and when a transition building is built). At those times, it requests and records values for the eight features shown in Table 2. This set balances information on recent game changes (i.e., the first two features), the opponent’s situation (e.g., Workers$_{op}$), and the player’s situation (e.g., Workers$_{p}$). When games begin, the value of the first two features is 0, while the others have small values. About 50 units are created, per side, in a short game, and a player’s limit is 200 at any time.

Cases are grouped by the building state, and at most one case is recorded per building state per game. In our experiments, games lasted an average of 5 minutes, and CAT made a maximum of 8 decisions in that time (see Figure 1); therefore, CAT does not require a fast indexing strategy.

The similarity between a stored case $C$ and the current game state $S$ is defined as:

$$\text{Sim}_{C,S} = (C_{\text{Performance}} / \text{dist}(C_{\text{Description}}, S)) - \text{dist}(C_{\text{Description}}, S)$$

where dist() is the (unweighted, unnormalized) Euclidean distance between two cases for the eight features. We chose this simple function because it emphasizes distance, and it prefers the higher-performing of two equally distant cases. It is particularly useful for building state 1 (i.e., the start of the game), when all case descriptions are all identical.

CAT uses a modified k-nearest neighbor function to select case Tactics for retrieval. Among the k most similar cases, it retrieves one with the highest Performance. However, to gain experience with all tactics in a state, case retrieval is not performed until each available Tactic at that state is selected $e$ times, where $e$ is CAT’s exploration parameter. During exploration, one of the least frequently used tactics is retrieved for reuse. Exploration also takes place whenever the highest performance among the k-nearest neighbors is below 0.5. In our experiments, $e$ and $k$ were both set to 3; we have not attempted to tune these parameters.

4.2 Reuse

CAT’s reuse stage is given a retrieved tactic. While adaptation takes place, it is not controlled by CAT, but is instead performed at the level of the action primitives in the context of the Wargus game engine (e.g., if an action requests the creation of a building, the game engine decides its location and which workers will construct it, which can differ in each game situation).

4.3 Revision

Revision involves executing the reused tactic in Wargus, and evaluating the results. No repairs are made to these tactics; they are treated as black boxes.

Evaluation yields the performance of a case’s tactic, which is a function of the increase in the player’s Wargus game score relative to the opponent. The score is measured both when the tactic is selected and at the game’s end, which occurs when one player eliminates all of the opponent’s units, or when we terminate a game if no winner has emerged after ten minutes of clock time.
The performance for the tactic of case $C$ for building state $b$ is computed as:

$$\text{Won}_i = \begin{cases} 1, & \text{if } \text{CaT wins} \\ 0, & \text{if } \text{CaT loses} \end{cases}$$

$$C_{\text{Performance}} = \sum_{i=1,n} C_{\text{Performance},i}/n$$

$$\Delta \text{Score}_i = 1/3 (\Delta \text{Score}_{i,p} + \Delta \text{Score}_{i,o} + \text{Won}_i)$$

$$\Delta \text{Score}_{i,p} = (\text{Score}_{i,p}\text{'s score after } C \text{'}s tactic executes in game } i) - (\text{Score}_{i,p,b} \text{ before } C \text{'}s tactic executes in game } i)$$

$$\Delta \text{Score}_{i,o} = (\text{Score}_{i,o}\text{'s score after } C \text{'}s tactic executes in game } i) - (\text{Score}_{i,o,b} \text{ before } C \text{'}s tactic executes in game } i)$$

$$\text{Sim}(C,S) = \frac{C_{\text{Performance}} - \text{dist}(C_{\text{Description}}, S)}{\text{dist}(C_{\text{Description}}, S)}$$

where $n$ is the number of games in which $C$ was selected, $\text{Score}_{i,p}$ is the player’s $Wargus$ score at the end of the $i^{th}$ game in which $C$ is used, $\text{Score}_{i,o,b}$ is player $p$’s score before $C$’s tactic is executed in game $i$, and $\text{Score}_{i,o,b+1}$ is $p$’s score after $C$’s tactic executes (and the next state begins). Similarly, $\text{Score}_{i,o}$ is the opponent’s score at the end of the $i^{th}$ game in which $C$ is used, etc. Thus, $C$’s performance is updated after each game in which it is used, and equal weight is given to how well the player performs during its state and throughout the rest of the game.

### 4.4 Retention

During a game, CaT records a description when it enters each building state, along with the score and tactic selected. It also records the scores of each side when the game ends, along with who won (neither player wins a tie). For each building state traversed, CaT checks to see whether a case $C$ exists with the same $<$Description, Tactic$>$ pair. If so, it updates $C$’s value for Performance. Otherwise, CaT creates a new case $C$ for that BuildingState, Description, Tactic, and Performance as computed in Section 4.3 (this counts as $C$’s first application). Thus, while duplicate cases are not created, CaT liberally creates new ones, and does not employ any case deletion policy.

### 5 Evaluation and Analysis

Our evaluation focuses on examining the hypothesis that CaT’s method for selecting tactics outperforms the best of the eight counter-strategies.

#### 5.1 Competitors: WARGUS Players

CaT’s performance is compared against the best performing counter-strategies. We would like CaT to outperform the best counter-strategies in terms of achieved score and frequency of victory against the opponents listed in Table 1. CaT has the advantage over the static evolved strategies, in that it can adapt its strategy to the opponents based on the three sources of domain knowledge described in Section 3. However, the best evolved counter-strategies achieve high results against the opponents and are tough to outperform.

#### 5.2 TIELT Integration

TIELT (Aha & Molineaux, 2004) is a freely available tool (http://nrlsat.ittid.com) that facilitates the integration of decision systems and simulators. We used this tool to perform the experiments described. The Game Model and Game Interface Model came from the Wargus integration led by Lehigh University (Ponsen et al., 2005b). We developed an agent to isolate the decision tasks to be performed and interpret the decisions of the Decision System; we also developed a Decision System Interface Model which describes communications between the CaT system and TIELT. Finally, the experiment methodology describes the data to be collected and the procedure for running the experiment described in the next section.

### 5.3 Empirical Methodology

We compared CaT to the evolved counter-strategies for its ability to win $Wargus$ games. We used a fixed initial scenario, on a 128x128 tile map, involving a single opponent. The opponent used one of the eight strategies listed in Table 1. With the exception of the opponent strategies designed by students, these were all used in (Ponsen and Spronck, 2004).

We chose the percentage of the total score as defined by $Wargus$, and the frequency of wins against a test opponent as our dependent variables. Because the environment is non-deterministic (due to communications latencies), we averaged the difference in score over ten trials, and measured the percentage of wins over the same period.

We performed an eight-fold cross validation study, training CaT against seven of the available opponents, and measuring performance against the eighth. For each training trial, the opponent was randomly selected from among the seven training opponents; after every 25 training trials, CaT was tested against the eighth opponent for ten trials.

### 5.4 Results

Figure 2 compares CaT’s frequency of victory to the average and best performance of the eight counter-strategies. On average, the eight counter-strategies win 34% of the time. Evolved_SC5 wins 72% of the time; this is the best frequency among the counter-strategies. After 100 trials, CaT wins 79% of the time. We compared the frequency of victory for CaT after 100 games against each opponent with Evolved_SC5’s performance against the same opponents, using a paired two sample one-tail t-test for means; the results were not statistically significant. We also compared the average score percentage against each opponent using the same test; the difference for this comparison was significant at the .05 level.

However, Evolved_SC5 was not the best performer in terms of score; Figure 3 compares CaT against Evolved_SR, the best scoring of the eight counter-strategies. CaT achieved 65% of the score on average, whereas Evolved_SR achieved 64%. This difference was not shown to be significant. However, Evolved_SR won only 49% of the time (it was the 3rd best on this metric). Here, CaT outperforms Evolved_SR at the .01 level.
This is the first attempt that we know of to learn to defeat an unknown opponent at a real-time strategy game using case-based reasoning. We were not able to significantly outperform all of the static counter-strategies, but we were able to perform at least as well as the best counter-strategies on two separate metrics, which no single counter-strategy could match.

6 Discussion

In Section 5, we showed that case-based techniques can be used to learn how to defeat an unknown opponent. The static tactics developed using Ponsen and Spronck’s genetic algorithm, although evolved to defeat a single opponent in a deterministic setting, were also somewhat flexible in the non-deterministic environment introduced by controlling WarGus externally. Although they no longer enjoyed a 100% success rate against the opponent that each was evolved to defeat, some of their success rates were high in our experiments. CaT was able to combine the successes of individual counter-strategies, showing that case-based selection of tactics is a viable strategy.

More study could improve CaT’s performance in this experiment. One flaw in our methodology is that the various features of the case description are not normalized, giving some features a larger effect on the distance calculation. This should be investigated, and there may be a possibility of improvement.

No attempt has been made to calculate the optimality of the opponents or variance between them. The opponents we used may not cover the space of possible opponents well; some are certainly harder than others, and may require qualitatively different strategies to defeat. This is supported by the fact that some strategies are much more difficult for the CaT system to defeat without training on them directly. More opponents, that better cover the space of possible strategies, may provide better approximations for CaT’s learning rate on the entire strategy population by reducing the variance within the opponent pool.

The problem of tactic selection can be viewed as a reinforcement learning problem, and several reinforcement learning approaches have been used with tasks in the game of Stratagus (Guestrin et al., 2003; Marthi et al, 2005). Some similarities exist between CaT’s selection algorithm and a reinforcement learning approach, in that a performance value is used to guide future actions in similar states. However, CaT’s approach disregards optimized long-term rewards, while focusing more heavily on degrees of similarity than traditional reinforcement learning approaches.

7 Conclusion

This is the first attempt to use a case-based reasoning system to win real-time strategy (RTS) games against an unknown opponent. Our experiments show that CAT learns to perform as well as or better than the best performing counter-strategy on score and victory frequency metrics.

The TIET system was able to adapt an earlier experiment (Aha, Molineaux, and Ponsen 2005) quite easily to perform the new experiment we report in this paper. Ongoing research on this system is expected to benefit future work along these lines. Furthermore, our experience with the new experiment highlights key areas for expansion of TIET’s capabilities.

CAT’s algorithm has not been tailored for this application; its performance can probably be further improved. Also, many interesting research issues require further attention, such as CAT’s applicability to online learning tasks, and transferring learned knowledge to win other games.
Acknowledgements

This research was supported by DARPA’s Information Processing Technology Office and the Naval Research Laboratory.

References


