Case-Based Reasoning for Build Order in Real-Time Strategy Games

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

Abstract

We present a case-based reasoning technique for selecting build orders in a real-time strategy game. The case retrieval process generalizes features of the game state and selects cases using domain-specific recall methods, which perform exact matching on a subset of the case features. We demonstrate the performance of the technique by implementing it as a component of the integrated agent framework of McCoy and Mateas. Our results demonstrate that the technique outperforms nearest-neighbor retrieval when imperfect information is enforced in a real-time strategy game.

Introduction

Real-time strategy (RTS) games are becoming one of the most competitive genres of gaming. Several international competitions are held for RTS games (World Cyber Games, BlizzCon, IeSF Invitational) and South Korea has a professional league devoted to StarCraft. Competitive RTS play requires making strategic and tactical decisions under real-time constraints. At the strategic level, players make high-level decisions such as which technologies to research and how much to invest in setting up a strong economy versus producing combat units. At the tactical level, players decide when and where to engage opponents. Additionally, players perform constant reconnaissance in order to develop counter strategies.

A major challenge for RTS games is developing AI systems capable of playing at a competitive level. Buro (2003) argues that RTS games offer a large variety of fundamental research problems, including decision making under uncertainty, opponent modeling and learning, and adversarial real-time planning. Competitive players overcome these challenges by studying replays of professional matches and developing strategies and tactics to employ in future matches.

One of the focuses of strategic play in RTS games is build order. A build order defines the sequence in which buildings are constructed, units are produced and technologies are researched. Build orders target a specific strategy, such as rushing or timing attacks. Rush strategies attempt to overwhelm an opponent with inexpensive combat units early in the game, while timing attacks engage opponents based on a trigger, such as completing an upgrade. For a given matchup, only a subset of the possible build orders are viable options due to the topology of the map and playing style of the opponent. For example, a rush strategy is strong on small maps with a direct path to the opponent's base, but weak on maps with a large travel distance between bases.

RTS games enforce imperfect information through a "fog of war", which limits visibility to portions of the map where the player controls units. In order to acquire information about an opponent, it is necessary to actively scout the map to find out which buildings and units the opponent is producing. Scouting is vital in RTS games, because different strategies have different types of counter strategies.

We present a case-based reasoning system for selecting build orders in Wargus, a real-time strategy game. The system communicates with the integrated agent framework of McCoy and Mateas (2008) and plays complete games of Wargus. The performance of the system is evaluated on a variety of maps in perfect and imperfect information environments.

Related Work

Case-based reasoning has been applied to strategic and tactical aspects of RTS gameplay. Molineaux et al. (2008) combine case-based reasoning and reinforcement learning for selecting actions at the tactical level of gameplay.

Aha et al. (2005) use case-based reasoning for strategy selection. The system uses several forms of domain knowledge, including a building-specific state lattice developed by Ponsen et al. (2005). The lattice contains nodes which represent completed buildings and state transitions corresponding to constructing an additional building. The lattice also contains counter-strategies for each state. Aha et al. expand this representation by mapping game situations to specific strategies. The system performs nearest-neighbor retrieval using the Euclidian distance between eight features in case descriptions. Aha et al. demonstrate that the system is capable of defeating opponents randomly selected from a pool of scripted strategies.

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

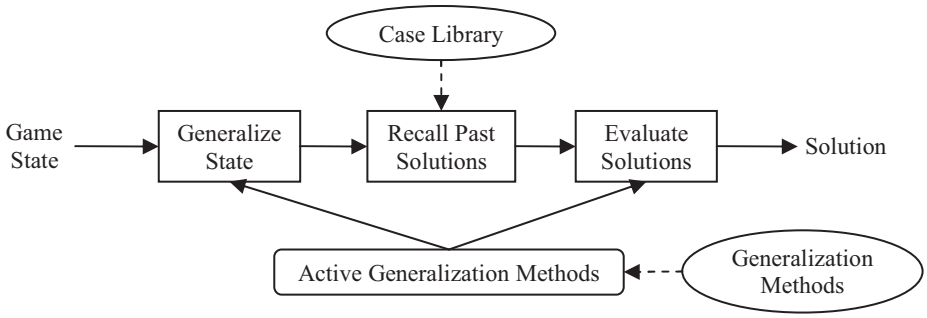

Figure 1 – Retrieval with conceptual neighborhoods

Ontañón et al. (2007) present a case-based planning approach for RTS games that interleaves planning and execution. The system requires players to annotate game traces with the goals being pursued for each action in the trace. Cases are extracted from the traces and specify primitive actions to perform or additional sub-goals for the current behavior. The system performs nearest neighbor retrieval using the game state and the current goal being pursued by the planner. They demonstrate that the performance of the system improves when additional game traces are available.

Limited work has focused specifically on build order. Kovarsky and Buro (2006) view build order as an optimization problem and apply planning to two problems: maximizing the number of units produced in a given period and producing a certain number of units as fast as possible. The system is suitable for developing build orders in an offline environment. Chan et al. (2007) present an online planner for build order and focus on resource gathering.

Recent work has explored the effects of enforcing imperfect information in RTS games. Bakkes et al. (2007) use TD-learning for evaluating tactical positions in Spring, a real-time strategy game. Their results show that an evaluation based on the unit count is effective in the early stages of the game, while an evaluation based on tactical positioning is more accurate in later stages of the game.

Case-Based Reasoning for Build Order

In this section we describe how case-based reasoning can be applied to build order in Wargus, a clone of the game Warcraft II which was developed by Blizzard Entertainment™.

Retrieval with Conceptual Neighborhoods

The case-based reasoning system performs retrieval using conceptual neighborhoods (Weber and Mateas 2009), which resembles the Transform-Recall-Adapt Methods in MINSTREL (Turner 1994). An overview of the process is shown in Figure 1. First, the transform step selects 0 to *n* generalization methods and applies them to the current game state, where *n* is the maximum number of generalizations allowed. Next, the recall step performs exact matching using a set of recall methods. Then the system evaluates the recalled cases by computing a distance metric based on the applied generalization methods. Next, a case is selected from the set of recalled cases using a weighted random selection. Finally, the behavior contained in the selected case is performed by the agent.

Case Representation

We define a case as a game state and behavior pair. Game state is encoded as six features, shown in Table 1.

Feature	Description
Player tech state	Bitset specifying if the agent has at least one of the following buildings: • blacksmith • lumber mill • stronghold • ogre mound
Enemy tech state	Bitset specifying if the opponent has at least one of the following buildings: • barracks • blacksmith • lumber mill • stronghold • ogre mound
Combat units	Total number of the agent's: • grunts • axe throwers • catapults • ogres
Worker units	Number of the agent's: • peons
Production buildings	Number of the agent's: • barracks
Map properties	• Distance to opponent • Open path to opponent

Table 1- Case features

The map properties feature is described by two properties. The first property specifies if there is a direct land route to the opponent's base. The second property specifies a qualitative distance (close, medium, far) to the opponent's base.

The case representation contains only a single feature that refers to the opponent, which describes the tech buildings the opponent possesses. Tech buildings are the most effective objects to scout in a RTS game, because they are stationary and reveal the opponent's strategy. Combat units and workers are mobile and therefore more difficult to accurately scout. Reducing the number of features that describe the opponent decreases noise when performing retrieval in an imperfect information environment.

Cases also contain a behavior, which specifies a build order action to execute. The build order actions used by the system are shown in Table 2. The left column of the table lists generic RTS build order actions, while the right column lists Wargus specific actions.

Build Action Type	Wargus Action
Train worker	• Build peon
Train combat unit	• Train grunt • Train axe thrower • Train catapult • Train ogre
Construct production building	• Build barracks
Construct tech building	• Build lumber mill • Build blacksmith • Upgrade stronghold • Build ogre mound
Research upgrade	• Melee attack • Melee armor • Range attack

Table 2 - Build order actions

Generalization Methods

Generalization methods transform the game state by flagging features as generalized. Game state features that have been generalized are ignored when performing matching with recall methods. The system includes a generalization method for each feature in the case representation, excluding the player tech feature. A generalization method for player tech state was not included, in order to constrain exploration of the case space.

Each generalization method has a corresponding edit distance, which computes the distance between the game state and a recalled case for the feature generalized. The edit distance computes the amount of in-game resources required to transform a recalled case to the game state. The system computes edit distance based on a linear combination of the gold and wood resources.

Generalize number of workers marks the number of workers feature as generalized. The edit distance is the cost of a worker unit times the difference in number of worker units between the game state and a recalled case.

Generalize number of production buildings marks the number of production buildings feature as generalized and computes the edit distance based on the difference in number of production buildings times the cost of a production building.

Generalize number of combat units marks the number of combat units feature as generalized and computes the distance based on the difference in combat units between the game state and recalled case. Although the number of combat units feature is an aggregation, the distance is computed based on the individual unit types. Therefore, the distance metric distinguishes between expensive and inexpensive units.

Generalize enemy tech computes a distance metric based on the difference in enemy tech buildings between the game state and recalled case. The distance metric sums the cost of the buildings that are different.

Generalize map property causes recall methods to ignore map properties when retrieving cases. The distance metric is a constant cost and is incurred if any of the map properties differ between the game state and a recalled case.

Recall Methods

Recall methods perform exact matching using a subset of the features. Features that are marked as generalized do not require an exact match. Recall methods match only on cases with the corresponding behavior. Therefore, each recall method utilizes a disjoint subset of the case library. This enables domain knowledge to be applied to the retrieval process in the form of behavior specific recall metrics and preconditions.

Each recall method has a set of preconditions that specify if the recall method can be applied to the current game situation. If the preconditions in a recall method fail, then the recall method will not be used for case retrieval. Recall preconditions are used to prevent the system from selecting invalid cases during retrieval, such as preventing retrieval of a case with the behavior "Train grunt" if the agent does not possess a barracks. Additional domain knowledge can also be specified in the preconditions. For example, the "Construct production building" recall method prevents the agent from constructing more than a single barracks, unless the agent already has a large number of worker units. This precondition check captures the domain knowledge that a player should only construct additional production buildings if the player has a sufficient economy to utilize additional production buildings.

The system contains a recall method for each build action type. The subsets of features evaluated by the recall methods are shown in Table 3. The following features require an exact match between the game state and a case: map properties, player tech, enemy tech and number of production buildings. The number of workers and number of combat units features require that the game state contains at least as many units as a case.

We selected the feature subsets for each recall method based on analyzing expert replays. For example, domain knowledge is demonstrated by the "Research upgrade" recall method, which matches against only the player tech

and number of combat units features. If the player possesses several combat units and the tech buildings required to research an upgrade, then researching the upgrade is a preferred action.

	PT	ET	NC	NW	PB	MP
Train worker	✓	✓	✓	✓		✓
Train combat unit	✓	✓		✓	✓	✓
Production building	✓	✓	✓	✓	✓	✓
Tech building	✓	✓		✓		✓
Research upgrade	✓		✓			

Table 3 - Feature subsets for recall methods. Features include player tech (PT), enemy tech (ET), number of combat units (NC), number of worker units (NW), number of production buildings (PB), and map properties (MP).

A total edit distance is computed for recalled cases. The distance is the summation of the edit distances of each generalization method applied to the game state. This summation includes only edit distances of features contained in the recall method's feature subset. Weights are computed from the total distance using an inverse distance relation, adjusted so the value at zero is finite.

Implementation

Our system extends the integrated agent framework of McCoy and Mateas (2008). The case-based reasoning system communicates with the framework using the blackboard pattern. McCoy and Mateas' agent was modified to produce buildings and units based on events posted to the blackboard. Reconnaissance capabilities were also added to the agent.

Agent Architecture

The agent consists of an ABL agent connected to the Wargus RTS engine. A behavior language (ABL) is a reactive planning language (Mateas and Stern 2002) and communicates with Wargus using a JNI interface. Different competencies in the agent communicate with each other through ABL's working memory, which acts as a blackboard. An overview of the agent architecture is shown in Figure 2.

The agent is composed of distinct managers, each of which is responsible for performing one or more subtasks. The Strategy Manager is responsible for high-level strategic decisions and focuses on build order. The Production Manager is responsible for producing units and buildings, based on messages generated by the Strategy Manager. The Tactics Manager decides when and where to engage the opponent.

The Strategy Manager was modified to communicate with the build order selector. Rather than choose which units to produce, the Strategy Manager adds a build order request to working memory. The build order selector polls for requests from working memory and performs case retrieval when a request is present. The build order selector then adds the chosen solution to working memory. Finally, the Strategy Manager passes the unit or building to produce to the Production Manager.

The agent implements minimal scouting behavior. Once the number of worker units is greater than a threshold value, the agent assigns a worker to scout the opponent's base. The scouting unit is given a single command to move to the starting location of the opponent's base. If the scouting unit is destroyed, then a new scouting unit is sent out. The maximum number of scouts is limited by a threshold value. The agent stops sending out scouting units once a combat unit has been trained.

Several additional modifications were made to the agent. The Production Manager was modified to automatically build farms when the difference between supply and demand is less than or equal to one. Also, the Tactics Manager was modified to attack when the agent has more combat units than the opponent. The Tactics Manager violates the imperfect information constraint, because it peeks at how many combat units the enemy possesses. However, this information is not utilized by the build order selector.

Build Order Selectors

Three case-based reasoning techniques were implemented for the build order selector component. The conceptual neighborhood selector (CNS) retrieves cases using the conceptual neighborhood approach discussed above. The nearest neighbor selector (NNS) performs retrieval using Manhattan distance based on the unit and building counts of all unit types, for both of the players. The random case selector (Random) selects build actions randomly from the set of currently valid cases.

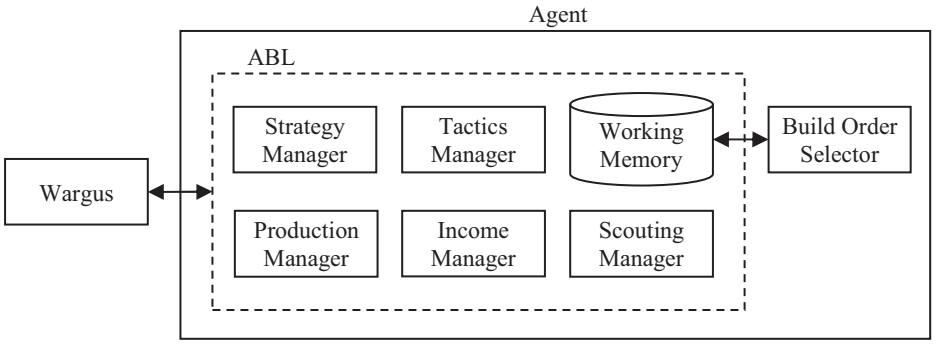

Figure 2 - Agent architecture

Case Library

A case library was generated by running several different scripted builds against each other on several maps. The scripts were selected from eight hand-coded build orders with specific timing attacks. The validation scripts discussed in the results section were not used to generate the case library.

The map pool consisted of maps with varying distances between bases (close, medium, far) and open and closed paths between bases. Four scripts were selected for each map and tested against the other scripts, for a total of six matches per map. Cases were added to the library only for the winning script. The case library consists of 36 game traces which contains over 1500 cases. The case library is much larger than previous work that utilizes game traces to build the case library (Ontañón et al. 2007; Mishra et al. 2008).

Empirical Study

The agent was tested against the built-in AI of Wargus, two well-established scripts and a new script. The land attack (LA) script is the default AI for Wargus and produces a large variety of ground forces. The soldiers rush (SR) performs a heavy melee attack early in the game. The knights rush (KR) performs a light melee attack and then upgrades to a stronghold and attacks the opponent with a heavy melee attack. The knights rush is considered a near-optimal Wargus strategy (Ponsen et al. 2005). The fast ogre (FO) script trains twelve workers and immediately upgrades to ogres.

	Perfect Information	Imperfect Information
Random	31%	19%
NNS	69%	50%
CNS	75%	66%

Table 4 - Win rates for build order selectors in perfect and imperfect information environments over 32 trials

The three build order selectors were tested against the scripted strategies. Four different maps were used for each matchup. The first three maps contain a direct land route to the opponent's base of varying distance (close, medium, far). The last map (NWTR) is a variation of the map "No-where to run, nowhere to hide". Ontañón et al. (2007) have focused their work on this map. Other researchers have focused on variations of medium sized maps with an open path to the opponent's base (Aha et al. 2005; McCoy and Mateas 2008; Ponsen et al. 2005). Two games were run on each map, resulting in a total of eight games per matchup.

	LA	SR	KR	FO	Overall
Random	62%	12%	0%	0%	19%
NNS	62%	62%	38%	38%	50%
CNS	100%	75%	50%	38%	66%

Table 5 - Win rates versus counter strategies over 8 trials

Matches were run with perfect and imperfect information and results are shown in Table 4. The conceptual neighborhood selector won 66% of games in an imperfect information environment and outperformed nearest neighbor retrieval. Also, the success rate of the conceptual neighborhood selector decreased by only 9% when enforcing imperfect information, while the success rate of the nearest neighbor selector decreased by 19%. Note that the performance of the random selector is worse in the imperfect information environment, because worker units are used for scouting in addition to resource gathering.

	Open Close	Open Medium	Open Far	NWTR
Random	25%	25%	25%	0%
NNS	62%	12%	88%	25%
CNS	75%	62%	75%	50%

Table 6 – Win rates on the map pool over 8 trials

Win rates against the scripted builds with imperfect information enforced are shown in Table 5. The conceptual neighborhood selector outperformed the other selectors in all matchups, while the random selector performed worst in all matchups. The conceptual neighborhood selector achieved a success rate of at least 50% on every map (see Table 6). These results indicate that the conceptual neighborhood approach was better at adapting to new game situations.

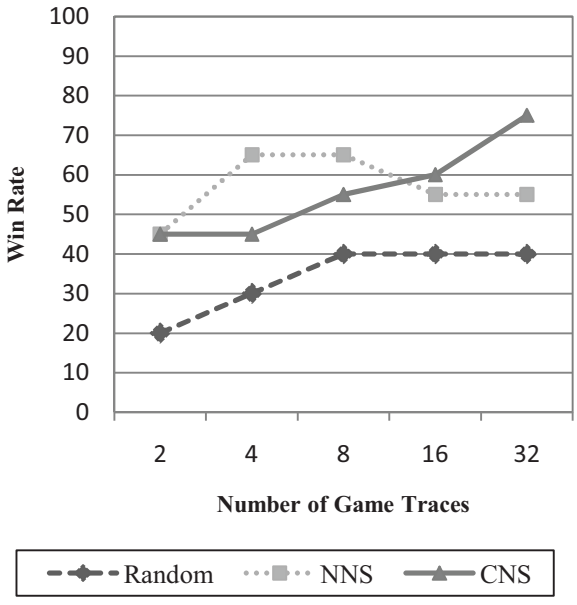

Figure 3 - Number of game traces versus win rate

The build order selectors were evaluated with different case library sizes. For each case library size, random subsets of traces were selected and each build order selector was tested over 20 trials. The results are shown in Figure 3. The nearest neighbor selector performed best

when the case library consisted of four or eight game traces, while the performance of the conceptual neighborhood selector continued to improve as more traces were added to the case library.

	Ponsen	Ontañón	McCoy	CNS
LA	76%	89%	-	100%
SR	29%	-	80%	75%
KR	13%	-	53%	50%

Table 7 - Reported success rates

Our results are compared to reported success rates from the literature in Table 7. Aha et al. (2005) report an average success rate of over 80%, but do not specify a win rate against the soldiers and knights rushes. All prior work, to our knowledge, has reported success rates using perfect information, while we are reporting success rates for an imperfect information environment.

Conclusion

In this paper we have demonstrated how conceptual neighborhoods can be applied to case-based reasoning for selecting build orders in a RTS game. Our contributions include applying conceptual neighborhoods to feature vectors in case-based reasoning, enforcing imperfect information in a RTS game, testing against scripts on a wide variety of maps, and performing proper validation using distinct training and validation script sets.

Our results show two interesting properties. First, the conceptual neighborhood approach achieved similar success rates to nearest neighbor retrieval in a perfect information environment, while outperforming nearest neighbor retrieval when imperfect information is enforced. This leads us to conclude that the conceptual neighborhood approach is better at adapting to new game situations. Second, the performance of the conceptual neighborhood approach improved when more traces were available, while the nearest neighbor approach was most effective when a small number of traces were available.

Future work will evaluate the performance of the system when utilizing a large collection of game traces.

References

Aha, D. W.; Molineaux, M.; and Ponsen, M. 2005. Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR)*, 5–20.

Bakkes, S.; Spronck, P.; van den Herik, J.; and Kerbusch, P. 2007. Predicting Success in an Imperfect-Information Game. In *Proceedings of the Computer Games Workshop (CGW)*, 219–230.

Buro, M. 2003. Real-time strategy games: A new AI research challenge. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1534–1535.

Chan, H.; Fern, A.; Ray, S; Wilson, N.; and Ventura, C. 2007. Online Planning for Resource Production in Real-Time Strategy Games. In *Proceedings of the International Conference on Automated Planning and Scheduling* (ICAPS), 65–72.

Kovarsky, A., and Buro, M. 2006. A First Look at Build-Order Optimization in Real-Time Strategy Games. In *Proceedings of the GameOn Conference*, 18–22.

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 AAAI Conference on Artificial Intelligence (AAAI)*, 1313–1318.

Mishra, K.; Ontañón, S.; and Ram, A. 2008. Situation Assessment for Plan Retrieval in Real-Time Strategy Games. In *Proceedings of the European Conference on Case-Based Reasoning (ECCBR)*, 355–369.

Molineaux, M.; Aha, D. W.; and Moore. P. 2008. Learning Continuous Action Models in a Real-Time Strategy Environment. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 257–262.

Ontañón, S.; Mishra, K; Sugandh, N.; and Ram, A. 2007. Case-Based Planning and Execution for Real-Time Strategy Games. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR)*, 164–178.

Ponsen, M.; Muñoz-Avila, H; Spronck, P.; and Aha, D. W. 2005. Automatically Acquiring Domain Knowledge for Adaptive Game AI Using Evolutionary Learning. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference (IAAI)*, 1535–1540.

Turner, S. 1994. *The Creative Process: A Computer Model of Storytelling and Creativity*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Weber, B. G. and Mateas, M. 2009. Conceptual Neighborhoods for Retrieval in Case-Based Reasoning. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR)*, 343–357.