Conceptual Neighborhoods for Retrieval in Case-Based Reasoning

Ben G. Weber and Michael Mateas

University of California, Santa Cruz
Santa Cruz, CA 95064, USA
{bweber, michaelm}@soe.ucsc.edu

Abstract. We present a case-based reasoning technique based on conceptual neighborhoods of cases. The system applies domain knowledge to the case retrieval process in the form of recall and generalize methods. Recall methods utilize domain specific preconditions and perform exact matching, while generalize methods apply transformations that generalize features in queries. The system uses a similarity function based on edit distances, where an edit distance considers only a subset of the features. This retrieval strategy enables the system to locate conceptually similar cases within the feature space. We demonstrate the performance of this approach by applying it to build-order selection in a real-time strategy game. Our results show that the system outperforms nearest neighbor retrieval when enforcing imperfect information in a real-time strategy game.

1 Introduction

One of the main challenges in case-based reasoning is constructing effective case representations and retrieval techniques. A common retrieval strategy involves using feature vectors for case descriptions and applying similarity functions for retrieval. The advantage of this approach is that techniques from the machine learning literature can be applied to case-based reasoning. However, the drawbacks of this approach are that it requires a comprehensive example set in order to achieve good results and is sensitive to noise [1].

Case retrieval can be improved by applying domain knowledge to case representation and retrieval. This can be achieved through the use of structural representations or deep features. Using symbolic representations enables case-based reasoning to integrate with other symbolic systems, such as planning. The main advantage of this approach is that a richer case representation enables domain specific retrieval and adaptation methods. The drawbacks of this approach are that it is computationally expensive and often requires annotated examples. Additionally, it can be difficult to encode spatial and temporal domain knowledge structurally.

We present a case-based reasoning technique that maps examples to conceptual neighborhoods of cases. The system uses a basic feature vector representation, but applies domain specific retrieval and adaptation methods. This enables domain knowledge to be described symbolically, rather than solely through feature vector weighting. This approach is suitable for knowledge rich domains that are difficult to represent structurally.

We apply conceptual neighborhoods to build-order selection in a real-time strategy (RTS) game. Build-order selection is a knowledge rich aspect of RTS games that incorporates temporal reasoning. Cases for build order can be extracted directly from game traces. The case-based reasoning system is integrated with the reactive planning agent of McCoy and Mateas [2]. Retrieval using conceptual neighborhoods is compared with variations of nearest neighbor while enforcing imperfect information in a RTS game.

The remainder of this paper is structured as follows: in the next section we discuss retrieval strategies for case-based reasoning. Section 3 introduces our approach to case retrieval using conceptual neighborhoods. We then apply conceptual neighborhoods to an RTS game in Section 4. Section 5 provides an overview of the system implementation and Section 6 reports our results. We compare our approach to previous work on case-based reasoning in RTS games in Section 7. Finally, we provide conclusions and future work in Section 8.

2 Retrieval in Case-Based Reasoning

Case retrieval selects cases based on a similarity metric. Nearest neighbor retrieval evaluates similarity by projecting cases in feature space and computing a distance between points. Structural approaches evaluate similarity by computing the number of transformations needed to translate cases to match a given query.

2.1 Nearest Neighbor Retrieval

Nearest neighbor is a form of instance-based learning [3] that has has been applied to classification problems and case retrieval in case-based reasoning systems. Nearest neighbor utilizes cases with a feature vector representation. Given a query, nearest neighbor retrieves the nearest case within the feature space. Similarity between cases is computed using a distance function.

The general form for computing the distance between a query, q, and a case, c, is defined by the L_p norm:

$$d(q,c) = \left(\sum_{j=1}^{n} |q_j - c_j|^p \right)^{1/p}$$

where q_j and c_j are features and n is the number features in the case description. This family of distance functions is also known as the Minkowski distance [4]. Common L_p norms are L_1, Manhattan distance, and L_2, Euclidean distance. Domain specific similarity functions can also be used for nearest neighbor retrieval. For example, edit distance, which computes the number of modifications needed to translate a case into the query, can be augmented with specific knowledge about data to produce knowledge-based similarity measures [5].

Nearest neighbor is sensitive to irrelevant and noisy features [1]. Variations of nearest neighbor have been developed to reduce these issues. Wettschereck and Aha [1] introduce a framework for automating the process of weighting features, which assigns low weights to irrelevant features. Bergmann and Vollrath show that a case can cover a region rather than a point in feature space [6]. They explore the use of generalized cases and present similarity functions for this representation. Another variation of nearest neighbor is neighborhood counting [7]. Wang defines neighborhoods as regions in feature space. To measure the distance between two data points, the similarity function computes the number of neighborhoods that cover a case and the query.

2.2 Structural Retrieval

Structural cases provide richer representations for case-based reasoning. Cases are commonly encoded as graphs, where the concepts of the problem domain are represented as nodes and relations between concepts are represented as edges. Edges can represent spatial, temporal, or causal relationships between nodes. Bunke and Messmer [8] introduce a similarity measure based on a weighted graph edit distance.

Structural representations enable case-based reasoning systems to perform problem solving, rather than just classification. For example, MINSTREL is an author-modeling story generator [9] that uses symbolic case-based reasoning. The system has a knowledge base of King Arthur stories and general knowledge about characters in this domain. MINSTREL's case representation enables the system to invent new stories that satisfy character and story goals.

Problem solving in MINSTREL uses knowledge representations known as Transform-Recall-Adapt Methods (TRAM). TRAMs are bundled knowledge representations that know how to transform a problem into a related problem, recall a solution to the new problem and adapt the solution back to the original problem. An example TRAM is Cross-Domain-Solution, which ontologically maps a problem into a new domain, solves the problem in that domain and adapts the solution by reversing the mapping. MINSTREL also uses TRAMs recursively. When performing recursive problem solving, MINSTREL transforms the original problem with multiple TRAMs, recalls a solution to the new problem and applies the adaptation step of the TRAMs to the recalled solution.

3 Conceptual Neighborhoods

We present a case-based reasoning system based on conceptual neighborhoods. The system is a hybrid approach between nearest neighbor retrieval and symbolic case-based reasoning. It shares with nearest neighbor methods the use of feature vector representations, while sharing with symbolic case-based reasoning the use of domain specific transform and recall rules.

Conceptual neighborhoods provide a way to organize cases using deep features. Representations based on conceptual neighborhoods have been shown to allow for reasoning on imprecise knowledge [10]. Conceptual neighborhoods have been applied to case-based reasoning in the legal domain. Hypo [11] uses claim lattices to represent conceptual neighborhoods of cases and exploit connections among cases relevant to the current problem. Conceptual neighborhoods can also be applied to case-based reasoning systems that use a feature vector representation by mapping surface features to deep features. Generalizing a deep feature in this representation enables exploration of a neighborhood of cases.

Our approach maps features to concepts and projects cases in concept space. Concepts can be composed of several features, resulting in a dimensionality reduction. This process is similar to systems that map surface features to deep or knowledge-intensive features [12]. The goals of this mapping are to reduce the effects of noise and enable generalization for case retrieval.

3.1 Case Retrieval

Case-based reasoning with conceptual neighborhoods resembles problem solving using TRAMs [9]. However, our approach differs from MINSTREL in that our system does not bound transformations to specific recall methods. An overview of the process is shown in Figure 1. First, the transform step selects 0 to n generalize methods and applies them to the query, where n is the maximum number of generalizations allowed. Next, the recall step performs matching using a set of recall methods. Then the system evaluates the recalled cases by computing a distance metric based on the applied generalize methods. Finally, a case is selected from the set of recalled cases.

Recall methods perform exact matching using a subset of the concepts. Concepts that are marked as generalized do not require an exact match, but incur a cost based on a distance metric. The subset of concepts to select is domain specific and is derived from domain knowledge. Matching functions can test for equivalence, greater-than or less-than relations or a domain specific matching function. Recall methods match only on cases with the corresponding class or behavior. Therefore, each recall method matches against a disjoint subset of the case library. Recall methods contain preconditions, which verify that retrieved

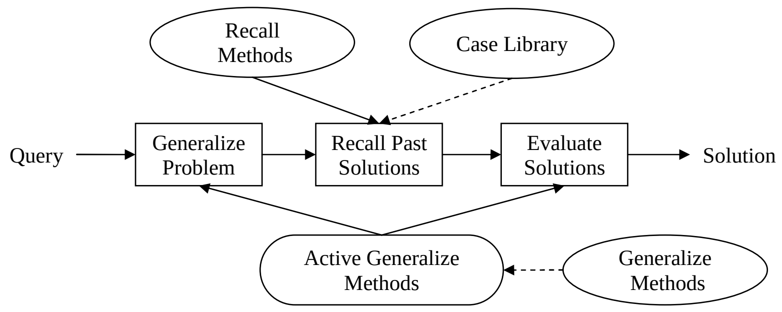

Fig. 1. Retrieval with conceptual neighborhoods

cases are valid given the query. Preconditions for recall methods can specify additional domain knowledge to improve recall performance. The use of recall methods enables the system to evaluate feature subsets based on possible solutions, which differs from previous work [12] that selects feature subsets based on the problem.

Generalize methods transform the query by flagging an individual concept in the query as generalized. There is a generalize method for each concept in the query that can be generalized. Generalize methods enable the system to search the problem space and solution space [9].

The evaluation step computes a distance metric for a case by summing the distance metrics of each generalize method applied to the case. Generalize methods compute an edit distance [8] for the generalized concept between the query and recalled case. The edit distance can be based on individual features mapped to the concept. The distance is zero if the generalized concept is not contained in the subset of concepts used by the recall method that selected the case. The evaluation step then selects a case using a selection strategy, such as highest similarity or weighted random selection.

3.2 Retrieval in Concept Space

The conceptual neighborhood approach maps features to concepts and retrieves cases in concept space. An example query is shown in Figure 2. The first graph (a) shows the query, s, and three cases. The second graph (b) demonstrates retrieval using nearest neighbor. The distance for retrieving c_1 using nearest neighbor is computed as follows:

$$d = \sqrt{(c_{1A} - s_A)^2 + (c_{1B} - s_B)^2}$$

The remaining graphs demonstrate retrieval using conceptual neighborhoods. The steps to retrieve c_1 in the third graph (c) are the following:

1. Recall fails at r_1, because $c_{1A} \neq s_A$
2. Generalize method g_A generalizes concept A
3. Recall fails at r_2, because $c_{1B} \neq s_B$
4. Generalize method g_B generalizes concept B
5. Recall succeeds at r_3
6. $d = distance(s_A, c_{1A}) + distance(s_B, c_{1B})$

where $distance(s_j, c_j)$ is a domain specific edit distance. The steps to retrieve c_2 in the fourth graph (d) are the following:

1. Recall fails at r_1, because $c_{2A} \neq s_A$
2. Generalize method g_A generalizes concept A
3. Recall succeeds at r_2, because the recall method for c_2 does not consider concept B
4. $d = distance(s_A, c_{2A})$

Note that different recall methods were used to retrieve c_1 and c_2. The recall method used to retrieve c_1 matched against both concepts, while the recall method used to retrieve c_2 matched against only concept A.

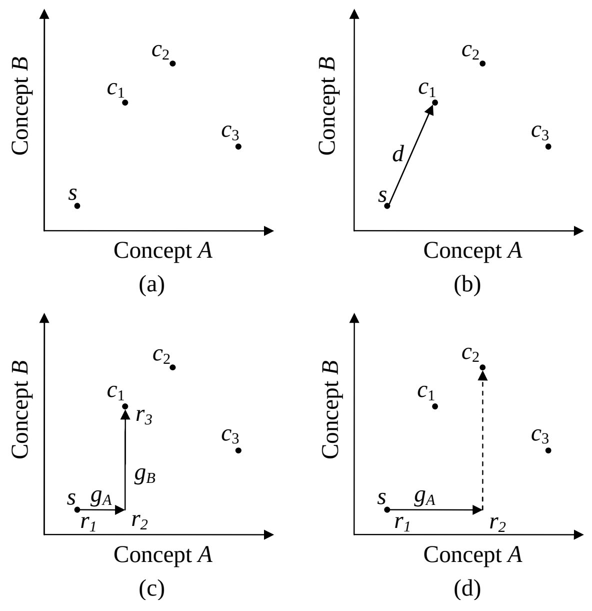

Fig. 2. Retrieval in concept space (a) The game state, s, and three cases (b) Retrieving c_1 using nearest neighbor (c) Retrieving c_1 using conceptual neighborhoods (d) Retrieving c_2 using conceptual neighborhoods

3.3 Applying Conceptual Neighborhoods

Conceptual neighborhoods can be applied to case-based reasoning systems that use a feature vector representation. The first step is to select a set of concepts and map the original features to concepts. The next step is to create a recall method for each class or behavior in the domain. The third step is to select concept subsets for each recall method. The final step is to select which concepts can be generalized and to determine edit distances for these concepts. This process is demonstrated in the next section.

4 Conceptual Neighborhoods in RTS Games

In this section we describe how conceptual neighborhoods can be applied to build order in Wargus[1], a clone of the game Warcraft II which was developed by Blizzard Entertainment^{TM}. The purpose of the case-based reasoner is to select the next unit or building to produce based on the current game state.

RTS games present a variety of research problems, including decision making under uncertainty, opponent modeling and adversarial planning [13]. RTS games enforce imperfect information through a “fog of war”, which limits visibility

[1] http://wargus.sourceforge.net

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.

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 order is a knowledge-rich aspect of RTS gameplay and players can improve their skills by studying replays of professional matches and learning which build orders work best against counter strategies on a variety maps.

4.1 Case Representation

We define a case as a behavior and game state pair. Behaviors are discussed in more detail in the following section. Game state includes the following concepts: player technological state (player tech), enemy technological state (enemy tech), number of combat units, number of workers, number of production buildings and map properties. The mapping of features to concepts is shown in Table 1.

4.2 Recall Methods

The build order behaviors in Wargus can be classified by the following actions: train worker unit, train combat unit, build tech building, build production building and research upgrade. The system contains a recall method for each behavior type. The subsets of concepts evaluated by the recall methods are shown in Table 2. The following concepts require an exact match between the query and a case: map properties, player tech, enemy tech and number of production buildings. The number of workers and number of combat units concepts require that the query contains at least as many units as a case.

We selected the concept subsets for each recall method based on analyzing expert replays. Domain knowledge is demonstrated by the research-upgrade recall method, which matches against only the player tech and number of combat unit concepts. If the player possesses several combat units and the tech buildings required to research an upgrade, then researching the upgrade is a preferred action.

Table 1. Game state features mapped to concepts

Concept	Features
Player Tech	Lumber Mill, Blacksmith, Stronghold, Mound
Enemy Tech	Enemy Barracks, Lumber Mill, Blacksmith, Stronghold, Mound
Prod. Buildings	Barracks
Workers Units	Peons
Combat Units	Grunts, Axe throwers, Catapults, Ogres
Map Properties	Distance to opponent base, Open path to opponent base

Table 2. Concept subsets for recall methods

	Map properties	Player tech	Enemy tech	Worker units	Combat units	Production buildings
Train worker	✓	✓	✓	✓	✓	
Train combat unit	✓	✓	✓	✓		✓
Tech building	✓	✓	✓	✓		
Production building	✓	✓	✓	✓	✓	✓
Research upgrade		✓			✓	

4.3 Generalize Methods

The system includes a generalize method for each concept in the case representation, excluding the player tech concept. Generalize methods mark a concept as not requiring an exact match at the cost of an edit distance. The edit distance computes the distance between the query and recalled cases based on the amount of in-game resources required to translate the query to a recalled case for a particular concept, by computing a linear combination of the gold and wood resources.

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

Generalize number of production buildings marks the number of production buildings concept 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 concept as generalized and computes the distance based on the difference in combat units between the query and recalled case. Although the number of combat units concept 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 query 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 query and a recalled case.

4.4 Case Selection

A case is selected from the set of retrieved cases using weighted random selection. Weights are computed using an inverse distance relation:

$$weight = \frac{1}{\delta + \sum_{j=1}^{n} distance_j(q_j, c_j)}$$

where q_j is a concept of the query, c_j is a concept of a case, $distance_j$ is the edit distance for concept j, n is the number of concepts and δ is a constant to adjust so the value is finite if the edit distance is zero.

A randomized selection is used to enable constrained exploration of the case library. This leads to small variations in build order. Also, always picking the best case can cause problems in an imperfect information environment, because noise can cause the similarity function to be inaccurate. The system uses an inverse distance relation, but other approaches could be used, such as exponential weighting.

4.5 Retrieval Example

An example query with three cases is shown in Figure 3. The cases correspond to training a peon (c_1), building a blacksmith (c_2) and building a barracks (c_3). In Wargus, the cost of a worker unit is 400 gold and the cost of a first-tier combat unit is 600 gold. Retrieving c_1 consists of the following steps:

1. Recall fails at r_1, because $c_{1w} \neq s_w$
2. g_w generalizes the number of worker units
3. Recall fails at r_2, because $c_{1c} \neq s_c$
4. g_c generalizes the number of combat units
5. Recall succeeds at r_3
6. $d_{c1} = distance(s_w, c_{1w}) + distance(s_c, c_{1c}) = 1 * 400 + 2 * 600 = 1600$

Retrieving c_2 consists of the following steps:

1. Recall fails at r_1, because $c_{2w} \neq s_w$
2. g_w generalizes the number of worker units
3. Recall succeeds at r_2
4. $d_{c2} = distance(s_w, c_{2w}) = 2 * 400 = 800$

Retrieving c_3 consists of the following steps:

1. Recall fails at r_1, because $c_{3w} \neq s_w$
2. g_w generalizes the number of worker units
3. Recall succeeds at r_2
4. $d_{c3} = distance(s_w, c_{3w}) = 3 * 400 = 1200$

Weights are computed based on these distances. Setting $\delta = 400$ results in the following weights: $w_{c1} = 0.0005$, $w_{c2} = 0.00083$ and $w_{c3} = 0.00063$. These weights are used to perform a weighted random selection.

5 Implementation

Our system uses the integrated agent framework of McCoy and Mateas [2]. 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. Six case retrieval strategies were implemented to evaluate the performance of conceptual neighborhoods.

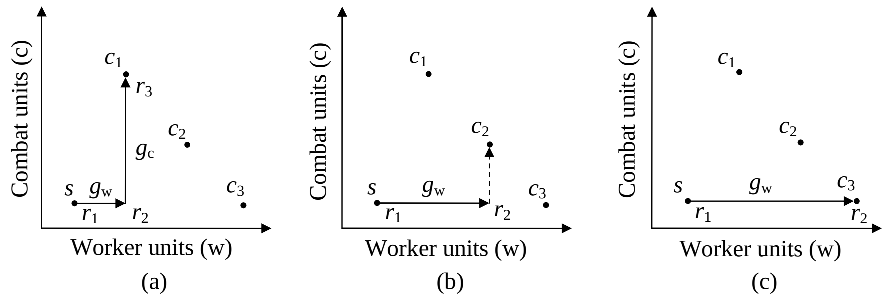

Fig. 3. An example query. The cases correspond to training a peon (a), building a blacksmith (b) and building a barracks (c).

5.1 Architecture

The game-playing agent consists of an ABL agent connected to the Wargus RTS engine. A behavior language (ABL) is a reactive planning language [14] and communicates with Wargus using JNI. Different competencies in the agent communicate with each other through ABL's working memory. ABL's working memory serves as a blackboard and enables communication through the blackboard pattern. An overview of the agent architecture is shown in Figure 4.

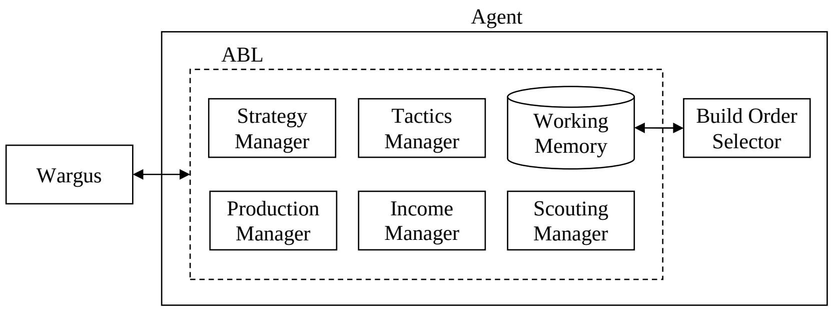

Fig. 4. Agent architecture

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 scouting manager is responsible for assigning worker units to scout the opponent base.

5.2 Build Order Selectors

Six retrieval strategies were implemented for the build order selector. The goal of implementing several retrieval strategies was to determine if retrieval with conceptual neighborhoods outperforms nearest neighbor and to evaluate if both generalize and recall methods are necessary for the conceptual neighborhood approach to be effective.

Random build order selector (Rand) picks cases randomly from the set of valid cases.

Nearest neighbor selector (NNS) performs case retrieval using Manhattan distance. The case description contains a feature for each unit type, for both players.

Case feature selector (CFS) performs case retrieval using the concepts and edit distances discussed in the previous section, but does not use generalize or recall methods.

Generalize methods only selector (GMS) performs exact matching using generalize methods. This approach does not use concept subsets for recall.

Recall methods only selector (RMS) performs partial matching using recall methods and concept subsets.

Conceptual neighborhood selector (CNS) uses both generalize and recall methods to perform case retrieval.

5.3 Case Generation

A case library was generated by running several different scripted builds against each other on several different 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 for case generation. 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 games per map. Cases were added to the library only for the winning script. The case library contains 36 games traces and over 1500 cases. The case library is much larger than previous work utilizing game traces [15, 16].

6 Results

The agent was evaluated against the built-in AI of Wargus, two well-established scripts and a new script. Four different maps were used, where the first three maps contain a direct land route to the opponent's base of varying distance (close, medium, far) and the last map (NWTR) is a variation of the map "Nowhere to run, nowhere to hide". The agent was tested in perfect and imperfect information environments. In games with perfect information, the game state is fully observable. In games with imperfect information, the "fog of war" is enforced, limiting the visibility of the agent to areas where units are controlled.

Table 3. Win rates for perfect and imperfect information environments over 32 trials

	Perfect Information	Imperfect Information
Rand	31%	19%
NNS	69%	50%
CFS	56%	44%
GMS	44%	41%
RMS	50%	47%
CNS	75%	66%

Games were run with perfect and imperfection information and results are shown in Table 3. The conceptual neighborhood selector won 66% of games in an imperfect information environment. 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%.

Table 4. Win rates versus scripted builds over 8 trials

	Land Attack	Soldier's Rush	Knight's Rush	Fast Ogre	Overall
Rand	62%	12%	0%	0%	19%
NNS	62%	62%	38%	38%	50%
CFS	100%	50%	12%	12%	44%
GMS	75%	50%	25%	12%	41%
RMS	88%	62%	25%	12%	47%
CNS	100%	75%	50%	38%	66%

Win rates against the scripted builds with imperfect information enforced are shown in Table 4. The conceptual neighborhood selector outperformed all of the other selectors. All of the retrieval strategies outperformed random selection, but the case feature, generalize methods only, and recall methods only selectors performed worse than nearest neighbor retrieval. The conceptual neighborhood selector achieved a success rate of at least 50% on every map (see Table 5). These results indicate that the conceptual neighborhood approach was better at adapting to new game situations.

7 Related Work

Case-based reasoning has been applied to several aspects of RTS play, including strategic [17] and tactical [18] levels of gameplay. There are two approaches that have been applied to case-based reasoning in RTS games: bootstrap learning systems that rely on exploration of the state space and systems that utilize game traces to automatically acquire knowledge about RTS gameplay.

Table 5. Win rates on the map pool over 8 trials

	Open Close	Open Medium	Open Far	NWTR
Rand	25%	25%	25%	0%
NNS	62%	12%	88%	25%
CFS	38%	50%	50%	38%
GMS	50%	12%	62%	38%
RMS	50%	50%	62%	25%
CNS	75%	62%	75%	50%

Aha et al. [17] use case-based reasoning to defeat strategies randomly selected from a pool of fixed strategies. The system uses the building-specific state lattice developed by Ponsen et al. [19], which abstracts Wargus game states into a state lattice and specifies a set of counter strategies for each state. This knowledge is used to explore the state space and build a case library of counter strategies. Our system differs in that a state lattice is not used to constrain the set of possible strategies.

Molineaux et al. [18] apply case-based reasoning to tactical situations in RTS games. They claim that tactical gameplay in RTS games is knowledge poor and therefore a state-space taxonomy is insufficient to encompass all relevant tactical decisions. They combine case-based reasoning and reinforcement learning in order to explore the state space. Our system varies from this approach, because our system makes use of domain knowledge.

Game traces have been used by case-based reasoning systems to play full RTS games. Ontañón et al. [15] present a case-based planning approach that uses game traces to interleave planning and execution, and play at the same action granularity as a human player. The system uses expert-annotated game traces to automatically acquire domain knowledge from expert players. Cases are extracted from traces and specify primitive actions or additional subgoals for the current behavior to pursue. Our approach differs in that we define a clear division between planning and case-based reasoning and case-based reasoning is applied only to build order.

Mishra et al. [12] extend the case-based planner by introducing situation assessment for improved case retrieval. They noticed that the performance of previous systems [15, 16] suffers when the case library stores numerous plans representing several strategies played over maps of different sizes. Mishra et al. introduce the concept of a situation and use situation assessment to aid in case retrieval. A situation is defined by a high-level representation of the game state including map properties and current goals of the player. Situations are then used to select relevant features for case retrieval. The main difference between this approach are our system is that Mishra et al. select feature subsets based on the current game state, while the conceptual neighborhood approach selects feature subsets based on the type of case being recalled. Additionally, previous work [15, 16, 12] has investigated smaller numbers of game traces, which also required annotation.

Table 6. Reported win rates versus the conceptual neighborhood selector with perfect information (PI) and imperfect information (II).

	Ponsen et al.	Ontañón et al.	McCoy& Mateas	CNS PI	CNS II
Land Attack	76%	89%	—	100%	100%
Soldier's Rush	29%	—	80%	75%	75%
Knight's Rush	13%	—	53%	50%	50%

Our results are compared to reported success rates from the literature in Table 6. All prior work, to our knowledge, has used perfect information. We report results for the performance of the system in perfect and imperfect information environments, which achieved the same win rates against the standard scripts. Aha et al. [17] report an average success rate of over 80%, but do not specify win rates against the soldier's and knight's rushes.

8 Conclusion

In this paper we have demonstrated how conceptual neighborhoods can be applied to retrieval in case-based reasoning. Our contributions include the application of conceptual neighborhoods to retrieval, which enables additional domain knowledge to be applied to case retrieval, and evaluation of conceptual neighborhoods versus other retrieval strategies. We validated our approach by applying conceptual neighborhoods to build order in a RTS game. The results indicate that retrieval using conceptual neighborhoods outperforms nearest neighbor when enforcing imperfect information.

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 approaches that did not use both generalize and recall methods performed worse than nearest neighbor retrieval. If the generalize methods only selector had outperformed nearest neighbor, then the success of the system could be attributed to the use of domain specific distance metrics. However, our results show that the use of concept subsets was necessary to improve retrieval. This indicates that the interaction between generalize and recall methods is necessary to capture domain knowledge for retrieval with conceptual neighborhoods.

Future work will explore different types of transformations and the application of conceptual neighborhoods to additional aspects of case-based reasoning, including case adaptation and on-line learning.

References

1. Wettschereck, D., Aha, D.: Weighting features. Lecture notes in computer science **1010** (1995) 347–358

2. McCoy, J., Mateas, M.: An Integrated Agent for Playing Real-Time Strategy Games. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, Chicago, Illinois, AAAI Press (2008) 1313–1318
3. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. Machine Learning **6**(1) (1991) 37–66
4. Bagherjeiran, A., Eick, C.F.: Distance function learning for supervised similarity assessment. In: Case-Based Reasoning on Images and Signals. Springer (2008) 91–126
5. Cunningham, P.: A taxonomy of similarity mechanisms for case-based reasoning. IEEE Transactions on Knowledge and Data Engineering (Forthcoming)
6. Bergmann, R., Vollrath, I.: Generalized Cases: Representation and Steps Towards Efficient Similarity Assessment. Lecture notes in computer science **1701** (1999) 195–206
7. Wang, H.: Nearest Neighbors by Neighborhood Counting. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(6) (2006) 942–953
8. Bunke, H., Messmer, B.: Similarity Measures for Structured Representations. Lecture notes in computer science **837** (1993) 106–118
9. Turner, S.R.: The Creative Process: A Computer Model of Storytelling and Creativity. Lawrence Erlbaum Associates (1994)
10. Freksa, C.: Temporal Reasoning Based on Semi-Intervals. Artificial Intelligence **54**(1) (1992) 199–227
11. Ashley, K., Rissland, E.: A case-based approach to modeling legal expertise. IEEE Expert: Intelligent Systems and Their Applications **3**(3) (1988) 70–77
12. Mishra, K., Ontañón, S., Ram, A.: Situation assessment for plan retrieval in real-time strategy games. In: Proceedings of the Ninth European Conference on Case-Based Reasoning, Trier, Germany, Springer (2008) 355–369
13. Buro, M.: Real-Time Strategy Games: A New AI Research Challenge. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, Morgan Kaufmann (2003) 1534–1535
14. Mateas, M., Stern, A.: A Behavior Language for Story-Based Believable Agents. IEEE Intelligent Systems **17**(4) (2002) 39–47
15. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: Case-Based Planning and Execution for Real-Time Strategy Games. Lecture notes in computer science **4626** (2007) 164–178
16. Sugandh, N., Ontañón, S., Ram, A.: On-Line Case-Based Plan Adaptation for Real-Time Strategy Games. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, Chicago, Illinois, AAAI Press (2008) 702–707
17. Aha, D., Molineaux, M., Ponsen, M.: Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game. Lecture notes in computer science **3620** (2005) 5–20
18. Molineaux, M., Aha, D.W., Moore, P.: Learning Continuous Action Models in a Real-Time Strategy Environment. In: Proceedings of the Twenty-First Florida Artificial Intelligence Research Conference, Coconut Grove, Florida, AAAI Press (2008) 257–262
19. Ponsen, M.J.V., Muñoz-Avila, H., Spronck, P., Aha, D.W.: Automatically Acquiring Domain Knowledge For Adaptive Game AI Using Evolutionary Learning. In: Proceedings of the Twentieth National Conference on Artificial Intelligence, Pittsburgh, Pennsylvania, AAAI Press (2005) 1535–1540