Integrating Case-Based Reasoning with Reinforcement Learning for Real-Time Strategy Game Micromanagement

Stefan Wender and Ian Watson

The University of Auckland, Auckland, New Zealand
s.wender@cs.auckland.ac.nz, ian@cs.auckland.ac.nz

Abstract. This paper describes the conception of a hybrid Reinforcement Learning (RL) and Case-Based Reasoning (CBR) approach to managing combat units in strategy games. Both methods are combined into an AI agent that is evaluated by using the real-time strategy (RTS) computer game StarCraft as a test bed. The eventual aim of this approach is an AI agent that has the same actions and information at its disposal as a human player. As part of an experimental evaluation, the agent is tested in different scenarios using optimized algorithm parameters. The integration of CBR for memory management is shown to improve the speed of convergence to an optimal policy, while also enabling the agent to address a larger variety of problems when compared to simple RL. The agent manages to beat the built-in game AI and also outperforms a simple RL-only agent. An analysis of the evolution of the case-base shows how scenarios and algorithmic parameters influence agent performance and will serve as a foundation for future improvement to the hybrid CBR/RL approach.

1 Introduction

RTS games, such as StarCraft, provide a challenging test bed for AI research. They offer a polished environment that includes numerous properties that are interesting for AI research, such as imperfect information, spatial and temporal reasoning as well as learning and opponent modeling [1]. However, these characteristics also make RTS game environments very complex. Even sub-problems, such as the control of units in combat situations, can not be completely solved by brute force algorithms. Furthermore, deterministic approaches such as state machines and decision trees are unable to simulate human-level AI in this type of environment. For these reasons, we chose StarCraft as a test bed for a machine learning (ML) approach that tries to learn how to manage combat units on a tactical level ("micromanagement").

Micromanagement requires a large number of actions over a short amount of time. It requires very exact and prompt reactions to changes in the game environment. The problem involves concepts like damage avoidance, target selection and, on a higher, more tactical level, squad-level actions and unit formations

[2]. The requirements are made more difficult by the use of a commercial game that, while enabling complete access to its functionality, not always reacts in the most exact and prompt fashion. Due to these factors, research in this area often focuses on more high-level problems such as strategic decision making and planning [3] [4]. We propose, that the characteristics of micromanagement that we mention above make it a challenging yet rewarding test bed for ML approaches such as CBR which focuses heavily on the acquisition and reuse of knowledge.

In this paper we describe a hybrid ML approach to managing units in StarCraft at a micromanagement level. This approach is based on previous work [5], where we use RL in an agent in a simplified combat scenario. While StarCraft offers a very large and complex set of problems, that RL agent focuses on a small subset of the overall problem space and is limited to one simple scenario. In its limited area of application, the simple RL approach is very successful in learning how to beat the built-in game AI. The hybrid CBR/RL agent presented here uses a more generalized model that is able to handle a broader set of problems. Combining CBR with RL helps to offset shortcomings of our simple RL agent, while retaining key features of its performance. Our hybrid agent uses CBR for its memory management, RL for fitness adaption and an influence map(IM)/potential field [6] for the abstraction of spatial information. Eventually, we plan to create an agent that has the actions and information at its disposal to learn how to handle the tactical and reactive layers of the game at the same level a human player does.

2 Related Work

Using StarCraft as a test bed for game AI research has recently seen a great boost in popularity. There are annual competitions for computer agents ('bots') playing complete games at different conferences [7]. The bots competing in these competitions usually employ a number of different AI techniques for the different problems (strategy, tactics, resource management, etc.) of the game. [2] describes the architecture of *EISBot*, a reactive planning agent that is based on a conceptual partitioning of gameplay into areas of competence. Each area is in turn handled by a separate manager. [8] describes the use of heuristic search to simulate combat outcomes and control units accordingly. Since the StarCraft game environment lacks in speed and precision, the authors first create their own simulator, *SparCraft*, to evaluate their approach. Using this simulator and a modied version of alpha-beta pruning, they look for the best moves for a given conguration of units. In [9], the authors introduce both a variation of UCT search, *UCT Considering Durations*, and a greedy-search-based technique called *Portfolio Greedy Search* and evaluate them against each other with *Portfolio Greedy Search* performing the strongest. The authors also include *SparCraft*-based combat simulation in their *UAlbertaBot*, currently the strongest competition bot [7].

Different forms of both RL and CBR are popular techniques in game AI research, either separately or as a combination of both approaches. The reinforcement learning component in this paper is based on [5]. While the simple

RL approach there shows good results for the one specific scenario that is tested, the work presented in this paper improves most its aspects by integrating CBR and thus enabling the generalization to a much larger and more complex area of problems. [10] describes the implementation of a simple, specialized RL-based agent for micro-managing combat units similar to [5]. However, their chosen state representation is highly tailored towards the experiments that they run and the size of the state space limits the efficiency of the learning agent. An extension of the managed units beyond those used in the experiments does not seem possible without entirely redesigning the RL model.

Using only RL for learning diverse actions in a complex environment becomes quickly unfeasible due to the curse of dimensionality [11]. Therefore, the addition of CBR to manage the obtained knowledge and to retain information in the form of cases offers benefits both in terms of the amount of knowledge that is manageable and knowledge generalization. [12] describes the integration of CBR and RL in a continuous environment. Both state- and action-space are continuous and the agent has to learn effective movement strategies for units in a RTS game. This approach is unique in that other approaches discretize these spaces to enable machine learning. As a trade-off for working with a non-discretized model, the authors only look at movement component of the game from a meta-level perspective where actions are given to groups of units instead of individuals and no actions concerning attacks are given.

Influence maps or potential fields for the abstraction of spatial information have been used in a number of domains, including game AI, after initially being developed for robotics [6]. [13] use influence maps to produce *kiting* in units, also in StarCraft. *Kiting* is a hit-and-run movement that is similar to movement patterns that our agent learns for micromanagement. However, spatial reasoning only covers parts of the micromanagement problem. [14] describes the creation of a multi-agent bot that is able to play entire games of StarCraft using a number of different artificial potential fields. However some of the parts of the bot use non-potential field techniques such as rule-based systems and scripted behavior, further emphasizing the need to combine different ML techniques to address larger parts of the problem space in an RTS game.

To our knowledge, all of the previous research in the area of machine learning techniques used for learning and controlling micromanagement and tactical decisions in RTS in general and StarCraft in specific have one thing in common: none of them address the entire micromanagement problem in dynamic fashion using a homogenous approach. Either only parts of the problem are addressed dynamically (i.e. either tactical decisions or reactive control) while other parts are controlled by scripted expert knowledge encoded as decision trees, finite state machines or in a similar fashion. Often the micromanagement problem is also split into different tasks, which are then addressed individually using different approaches that require extensive interfaces to communicate between each other.

Our goal therefore is to create an agent that is able to learn how to control any part of the problem space, resulting in a bot that can control the game much like a human can.

3 Agent Architecture

Our agent uses a CBR-based memory which utilizes RL to learn the fitness of its case solutions. The model of the game world is based on two different case-bases for different levels of abstraction of the current game state. RL is used to update the value of unit actions. Those unit actions represent the case solutions.

3.1 Case-Based Reasoning Component

The CBR component reflects a general CBR cycle [15], however does not use all of the cycle's phases. Two separate case-bases are used to reflect two different levels of granularity within the game. The first case-base contains information on the overall game state at a certain point in time. This includes information on the number and types of unit in the game as well as information on the state of the surroundings in the form of an IM. The second case-base stores and administrates per-unit information.

Retrieval is a multi-tiered process. At first, cases matching the overall current game in terms of agent and opponent units exactly and the overall game environment state to a certain degree of similarity are retrieved. The game environment state is encoded as an IM that reflects the areas of influence of the AI agent units and of enemy units. Figure 1 shows an excerpt of the field representing enemy unit influence. This influence is based on the damage potential of

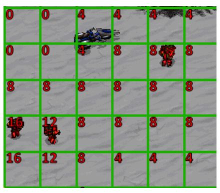

Fig. 1. Excerpt of the Influence Map

the units. Since an IM can theoretically be as big as a whole game map (up to 256*256 = 65536 fields), a suitable abstraction has do be found to compare similarities. We decided to use a histogram-based comparison [16]. This is possible if we regard an IM as a simple greyscale picture with intensity values representing the influence values. After converting IMs into histograms, we can use correlation to determine similarity. A nearest neighbor (NN) retrieval of cases in the case-base is done based on this similarity.

Cases from the game state case-base are linked to cases from the unit state case-base, i.e. each unit case is secondary to a game case (Figure 2). This results in a tree-like structure where each game state case branches out into a subset of unit state cases. Unit cases contain general information on units as well as a local IM, i.e. an influence map for the immediate surroundings of the unit. As

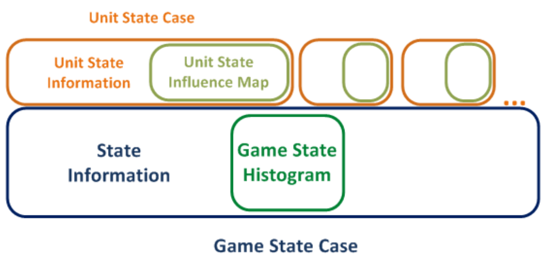

Fig. 2. Logical Structure of Cases and the Information they contain

this is the only precise spatial information the unit has, these unit-specific IMs can not be converted into histograms since that would invalidate the information they hold on positioning. Both general information and IM are used to compose a similarity score for comparison with the problem unit case.

The final step in the case retrieval process returns actions with assigned fitness values. These actions can then be **Reused** to solve the current problem case.

Currently, there is no **Revision** of case *descriptions* stored in the case-bases. Once a case has been stored, the case descriptions consisting of the general attributes and the IM that describe a game state or unit state are not changed anymore. The case *solution* that consists of actions and associated fitness values however, are adjusted each time they are selected and executed.

Retention is part of the retrieval process. In the first step of that process, if there is no sufficiently similar overall case, a new one is generated from the current environment state. Likewise, when searching the database for unit cases does not lead to any sufficiently similar results, a new unit case is created from the current problem.

3.2 Reinforcement Learning Component

The reinforcement learning component is used to learn fitness values for case solutions. This part of the agent is based on [5]. Because of this similarity, we chose to also use a Q-learning algorithm for policy evaluation. One-step Q-learning and Watkins' Q(λ), which uses eligibility traces, showed, in this specific setting, the best performance when compared to other tested algorithms by a small margin.

Q-learning is an off-policy temporal-difference (TD) algorithm that does not assign a value to states, but to state-action pairs [17]. Since Q-learning works independent of the policy being followed, the learned action value Q function directly approximates the optimal action-value function Q^*. Equation 1 shows the value update function for One-Step Q-learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right]. \quad (1)$$

This function is used to update the value of taking an action a_t in state s_t resulting in state s_{t+1}, given the learning rate α and a discount factor for future reward γ.

4 Model

Since RL is used to update the fitness values, the game environment has to be expressed in terms of a RL framework. The agent in a RL framework makes its decisions based on the state s an environment is in at any one time. If this state signal contains all the information of present and past sensations, it is said to have the *Markov property.* If a RL task presents the Markov property, it is a *Markov Decision Process* (*MDP*). A specific MDP is defined by a quadruple $(S, A, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$. RL problems are commonly modeled as MDPs, where S is the state space, A is the action space, $\mathcal{P}^a_{ss'}$ are the transition probabilities and $\mathcal{R}^a_{ss'}$ represents the expected reward, given a current state s, an action a and the next state s'. The MDP is used to maximize a cumulative reward by deriving an optimal policy π according to the given environment.

States States are represented by the cases of the CBR component minus the case solutions. Case architecture is explained in detail in section 3.1. A state is mostly specific to a given unit at a certain time, but also contains some information on the general game state. The unit-specific part of a state contains general information such as unit health, unit type and weapon cooldown. Both the general part and the unit-specific part contain an IM for spatial information. In the general game state part, this IM is abstracted into histograms. The local, unit-specific IM is made up of 7x7 tiles of the general IM and not abstracted.

Actions There are nine possible actions for the units in this model, one *Attack* action and eight *Move* actions. The *Move* actions are a compromise between freedom of action and limiting complexity, unit movement is abstracted to eight discrete directions, one for each 45°. The *Attack* action is handled by a very simple combat manager. The combat manager determines all enemy units within the agent's unit's range and selects the opponent with the lowest health.

Transition Probabilities While StarCraft has only few non-deterministic components, its overall complexity means that the transition rules between states are stochastic. As a result, different subsequent states s_{t+1} can be reached when taking the same action a_t in the same state s_t at different times in the game. There are no fixed time intervals, each unit will start and finish actions in a potentially different game frame. A unit transitions to a new state once its current action is finished.

Reward Signal Rewards, similar to cases, are computed on a per-unit basis. Rewards reflect the fitness value of an action that a certain unit took in a certain state. The reward signal is based on the difference in health of the agent unit in question, as well as the difference in health for enemy units it attacks. In other terms, this means the agent measures its success in the amount of damage it is able to deal while trying to retain as much of its own health as possible.

5 Empirical Evaluation and Results

There are three parts to our empirical evaluation. The first part, supplementary to the overall approach, is an optimization of algorithmic parameters for the

CBR and RL components. The second and most important part is evaluating the performance of our approach in different scenarios, compared both against the built-in game AI and our previous simple RL algorithm. In the third part, we also looked at the behavior of the case-bases that serve as the agent's memory during the performance evaluation. The motivation behind this is to identify trends pointing to bottle-necks in our approach and in order to find interesting behavior and points that require future improvement if we want to further extend the agent's abilities. We do this by recording and analyzing additional metadata for CBR. This includes the time that experimental runs take, as well as several metrics relating to action selection and case-base size. In order for RL to approximate an optimal policy p^*, there has to be a sufficient state- or state-action space coverage [18]. The amount of coverage that is sufficient in this context can be determined by evaluating at what point the agent is able to reliably achieve optimal results in terms of reward.

5.1 Experimental Setup and Parameter Selection

The agent is evaluated in two different scenarios. In Scenario A, the agent controls one fast, weak combat unit against six slower, stronger enemy units. This scenario is identical to the one that is used for the simple RL agent for 500- and 1000-game runs in [5], i.e. here the performance of the RL/CBR approach can be directly compared to simple RL. Scenario B is a variation of the first scenario that uses the same types of units. However, there are now three agent units and eight opponents. This scenario is targeted at evaluating both the effects and interactions of multiple units as well as a generally bigger scenario in comparison to the first, simplified one. Since the simple RL agent in [5] is only able to control one unit at a time, results for scenario B can not be compared to this, only against the built-in game AI.

In order to achieve the best performance possible, a first step is to optimize the parameters involved in both algorithmic components of our hybrid approach. A number of parameters are already decided through previous design decisions: Following the approach in [5], we decided to use Q-Learning as our RL algorithm. Both One-Step Q-Learning and Watkins' Q(λ) were tried and Watkins' Q(λ), as the better performing one, was eventually chosen for the performance evaluation. The retrieval method in the CBR component is a NN algorithm that uses histogram-based similarity. The similarity threshold during retrieval is not fixed and a set of different values is tested.

The optimal configuration of parameters was decided by running a large number of initial experiments and analyzing the results. Table 1 shows a summary of all variables selected for the scenarios as well as algorithmic parameters found through this initial evaluation. One change that was suggested by the first test runs was a change to the similarity metric used for the game state cases. These initial runs showed only very little learning success since the correlation similarity threshold that was used resulted in too many game states. As there is no reuse of unit cases that are grouped below different game cases, this leads to a state space that is too big to be sufficiently explored in reasonable time. For this reason, the similarity metric for the game state was adjusted to the point where game states are purely dependent on unit numbers.

The agent uses a declining ϵ-greedy exploration policy that starts with $\epsilon = 0.9$. This means, that the agent initially selects actions 90% random and 10% greedy, i.e. choosing the best known action. The exploration rate declines linearly towards 0 over the course of the 50, 500 or 000 games respectively.

Table 1. Evaluation Parameters

Parameter	**Values**
Number of Games	50, 500, 1000
Scenario	A(1vs6), B(3vs8)
Algorithm	Watkins' Q(λ)
CBR Similarity Threshold	60%, 80%,
RL Learning Rate α	0.2
RL Discount Factor γ	0.9
Trace Decay Rate	0.9
RL Exploration Rate ϵ	0.9 - 0

5.2 Performance

Having identified the optimal parameters, the next step is to run experiments using these parameters to analyze the agent's performance as a benchmark of the viability of our approach. In this step we run experiments of length 50 games, 500 games and 1000 games, each configuration 20 times. Shorter runs force the agent to earlier exploit its knowledge while longer runs allow a more extensive exploration of the state-action space. Figure 3, Figure 4 and Figure 5 show the results of running experiments on both scenarios with 50-game runs, 500-game runs and 1000-game runs. Since there was no statistically significant difference between results for the 60% similarity threshold and for the 80% similarity threshold, the diagrams display the results for 60% only, to improve readability. The average reward values are normalized to a range of 0% to 100% of the achievable score in a scenario, so the two scenarios can be compared with each other. Achieving 100% of the possible reward means, that the agent played a perfect game in which it destroyed all opposing units without sustaining any damage itself. The figures also show the results of random action selection which serves as a baseline for comparison. Furthermore, the results for scenario A are compared with results of the simple reinforcement learner in [5]. Since [5] uses a different model, there is a different initial minimum value for the reward. Reward development and overall reward are comparable though. The diagrams also include error bars that give a 95% confidence interval for the results.

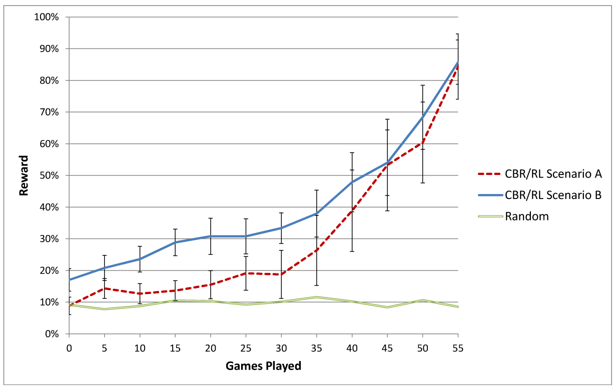

Fig. 3. Results for 50 Game Runs

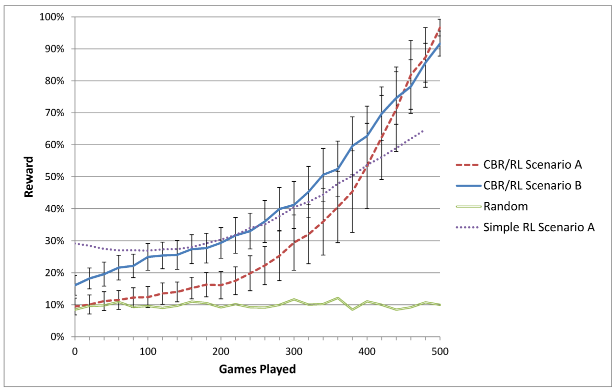

Fig. 4. Results for 500 Game Runs

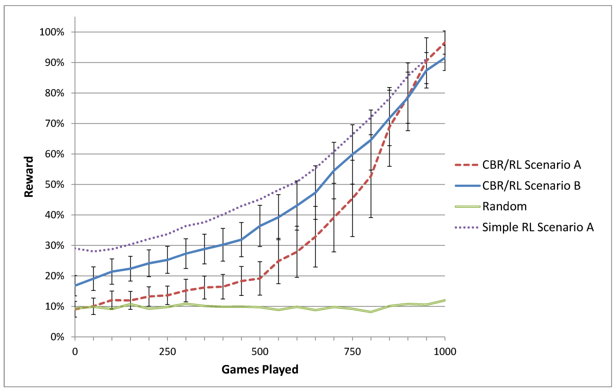

Fig. 5. Results for 1000 Game Runs

5.3 Case Base Development and State-Action Space Exploration

Table 2 shows the meta-data describing the development of the unit case-base for both scenarios. Since the number of game state cases is directly tied to the number of agent units, these cases are not separately listed. Scenario A has only one possible game state, Scenario B has three.

Table 2. Case Base Statistics

	Scenario A		**Scenario B**		
Similarity	**60%**	**80%**	**60%**	**80%**	**95%**
Number of Unit Cases:50 Games	2.26	7.3	6.63	101.8	1117.4
Number of Unit Cases:500 Games	2.3	11.1	7.8	204.67	2545.75
Number of Unit Cases:1000 Games	2.8	13.15	16.42	243.5	-
Visits per S/A Pair:50 Games	150.10	44.78	112.49	6.92	0.45
Visits per S/A Pair:500 Games	1264.20	252.15	806.93	32.83	0.04
Visits per S/A Pair:1000 Games	2041.49	441.34	1540.33	57.29	-
Unexplored S/A Pairs:50 Games	4.13%	26.33%	5.91%	52.90%	87.59%
Unexplored S/A Pairs:500 Games	2.90%	29.18%	2.35%	24.45%	85.24%
Unexplored S/A Pairs:1000 Games	2.58%	28.90%	1.69%	17.19%	-
Time Per Game in ms:50 Games	2716.11	2773.32	4218.20	7983.05	17559.11
Time Per Game in ms:500 Games	2678.18	2758.58	4506.65	7486.93	115486.5
Time Per Game in ms:1000 Games	2689.96	2726.64	4619.32	7729.05	-

The table also contains data from additional runs in Scenario B using a 95% similarity threshold. This threshold was introduced to see performance implications for the current model and algorithm when working with larger case-bases. The runs are not listed in the performance charts so as not to overload the diagrams. However, the obtained reward is considerably worse compared to lower similarity thresholds: only about 40% of the maximum reward in the 50 game runs and 70% for 500 game runs. The experimental run over 1000 games with a 95% similarity threshold could not be finished since, after about 600 games, a single game takes up to ten minutes to complete on a normal PC due to the amount of case comparisons required during each step.

6 Discussion

The figures show that the agent performs very well in both scenarios and is able to learn an optimal or near-optimal policy. The performance is far better than the random action selection baseline. Furthermore, the CBR/RL agent outperforms simple RL in 500-game runs in Scenario A where both can be compared. This suggests a faster speed of convergence to an optimal policy for the CBR/RL approach. Generally, the highest increase in reward obtained for both scenarios is in the second half of a run when exploration is slowly cut back and knowledge has already been obtained. Apart from a slightly more volatile development also shown by wider error bars, the policies learned within 50 games are basically on the same level as those of longer runs in terms of reward.

The average reward obtained by the CBR/RL agent over the course of one episode of 50, 500 or 1000 games follows a similar pattern for both scenarios. For Scenario A, the initial values are slightly higher than for Scenario B. This is potentially since three units that share the same memory are much faster at obtaining a basic level of knowledge than a single one. Eventually, the average reward for Scenario A becomes similar to that for Scenario B. When compared to the simple RL results, it is noticeable that the initial average reward for CBR/RL is lower. This is because of the different models used, especially the differences in action spaces. The model presented here has a much higher complexity than the one used in [5]. The initial average reward for simple RL is at the same level as random action selection in that model.

The data in Table 2 shows, that for 60% and 80% similarity thresholds, the number of unit cases in the case-base is relatively stable for experiments of different lengths, while varying considerably between similarities. The similarity in performance also means, that the number of unit cases in the case-base and thus the average number of visits to a state-action pair have a very wide range in which optimal results can be achieved. For Scenario B, the number of cases for different unit situations can range from just below 6 to more than 250.

An important metric for the performance of RL is the percentage of actions that are never explored. Since the RL methodology requires an infinite number of visits to each possible action or state-action pair to guarantee convergence, a large number of unexplored actions points to a potential problem. In this case, the performance did mostly not seem to be influenced in a negative way, despite some experiments having up to 50% unexplored actions. The performance only dropped significantly for runs with a 95% similarity threshold that resulted in more than 80% unexplored actions. However, even for experiments with a 80% similarity threshold where the performance is good, the amount of unexplored actions means that a large number of potentially good policies are never explored.

While the performance remains stable for similarity thresholds within a certain range, computational effort does not. The increased number of comparisons for retrieval resulting from the larger case-bases, leads to longer run times. For thresholds of 80% and especially 95%, running a larger number of games results in a large increase of unit cases. With lower thresholds the agent stops adding new unit cases at some point, and only explores the existing ones. For higher thresholds it keeps adding new unit states and new unexplored actions for much longer, potentially until the end of a run. This shows bad scaling and similarity thresholds that are too high for the given scenario, since cases are still being added despite greedy action selection. The behavior for a 60% similarity threshold, which is close to constant in terms of unit case numbers and time required, is ideal. For 80%, this is still partially true, however the number of unit cases does increase for longer runs, even though not in an extensive way.

The ideal similarity threshold depends on the complexity of the scenario. Both scenarios used in this evaluation are comparably simple, Scenario B slightly less than Scenario A. While it was not experienced in these experiments, it is to be expected that there is, depending on scenario complexity, a lower boundary for the similarity threshold beyond which cases are too general and learning is no longer possible. For these reasons, using a full scale combat scenario will require a careful selection of the similarity threshold.

7 Conclusion and Future Work

In this paper, we describe the integration of CBR and RL in an agent that controls units in combat situations in the RTS game StarCraft. The empirical evaluation demonstrates, that the model that is developed reflects the environment well, and that the agent scales appropriately when used with a larger scenario where the agent controls multiple units. The optimized algorithm parameters manage to obtain good results. The hybrid CBR/RL approach performs as well as or even better than our previous simple RL approach in the smaller scenarios where they are comparable, and manages to converge significantly faster towards a near-optimal policy.

As the eventual aim of our research is the creation of an agent that can handle the entire micromanagement problem, the analysis of the recorded meta-data gives valuable insights into the behavior of the current architecture of the CBR component and its potential for future developments. The current experiments work best with a low similarity threshold. Very high thresholds can lead to significant increases in terms of run-time and case-base size, while at the same time decreasing the agent's performance. However, since the number of cases also increases for more complex scenarios, a better handling of large scale case-bases is a problem that will have to be addressed in the future. When using the current approach in larger scenarios and eventually for the full micromanagement problem, higher similarity thresholds might also be required to distinctively separate between cases. Optimizing the case-retrieval method can improve the performance of the CBR component. Currently, a NN algorithm is used. However, with a larger case-base a more sophisticated method is needed to keep run times at reasonable levels, for example k-d trees to pre-index the cases [19].

Having shown the viability of hybrid CBR/RL, the next step in our research will be an increase in the complexity of the problem space by enabling the agent's full access to all StarCraft unit types and their associated actions and abilities. The next increment should also include the ability to manage groups of units at squad level. At the moment, the behavior of allied units is not taken into account at all in the reward computation to avoid diluting the effect of the units' own action. In the future we would like to change this to enable more team-oriented strategies. Since this is a significant increase in complexity, a further generalization might be required leading to a more generic CBR/RL architecture. Once our bot is able to handle this level of play, a comparison to the numerous existing competition bots, especially with the search-based micromanagement approach in *UAlbertaBot*, should provide an even better performance benchmark for hybrid CBR/RL RTS game micromanagement.

References

1. Buro, M., Furtak, T.: Rts games and real-time ai research. In: Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS), Citeseer (2004) 63–70
2. Weber, B., Mateas, M., Jhala, A.: Building human-level ai for real-time strategy games. In: 2011 AAAI Fall Symposium Series. (2011)
3. Weber, B., Mateas, M., Jhala, A.: Applying goal-driven autonomy to starcraft. In: Proceedings of the Sixth Conference on Artificial Intelligence and Interactive Digital Entertainment. (2010)
4. Churchill, D., Buro, M.: Build order optimization in starcraft. In: Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE 2011). (2011) 14–19
5. Wender, S., Watson, I.: Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar. In: Computational Intelligence and Games (CIG), 2012 IEEE Symposium on. (2012)
6. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. The international journal of robotics research **5**(1) (1986) 90–98
7. Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game ai research and competition in starcraft. (2013)
8. Churchill, D., Saffidine, A., Buro, M.: Fast heuristic search for rts game combat scenarios. In: AIIDE. (2012)
9. Churchill, D., Buro, M.: Portfolio greedy search and simulation for large-scale combat in starcraft. In: Computational Intelligence in Games (CIG), 2013 IEEE Conference on, IEEE (2013) 1–8
10. Micić, A., Arnarsson, D., Jónsson, V.: Developing game ai for the real-time strategy game starcraft. Technical report, Reykjavik University (2011)
11. Bellman, R.: Adaptive control processes: a guided tour. Volume 4. Princeton university press Princeton (1961)
12. Molineaux, M., Aha, D., Moore, P.: Learning continuous action models in a real-time strategy environment. In: Proceedings of the Twenty-First Annual Conference of the Florida Artificial Intelligence Research Society. (2008) 257–262
13. Uriarte, A., Ontañón, S.: Kiting in rts games using influence maps. In: Workshop Proceedings of the Eighth Artificial Intelligence and Interactive Digital Entertainment Conference. (2012)
14. Hagelbäck, J.: Multi-Agent Potential Field Based Architectures for Real-Time Strategy Game Bots. PhD thesis, Blekinge Institute of Technology (2012)
15. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI communications **7**(1) (1994) 39–59
16. Davoust, A., Floyd, M., Esfandiari, B.: Use of fuzzy histograms to model the spatial distribution of objects in case-based reasoning. In Bergler, S., ed.: Advances in Artificial Intelligence. Volume 5032 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 72–83
17. Watkins, C.: Learning from Delayed Rewards. PhD thesis, University of Cambridge, England (1989)
18. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
19. Wess, S., Althoff, K., Derwand, G.: Using k-d trees to improve the retrieval step in case-based reasoning. Topics in Case-Based Reasoning (1994) 167–181