Neuroevolution for Micromanagement in the Real-Time Strategy Game Starcraft: Brood War

Jacky Shunjie Zhen and Ian Watson

Department of Computer Science, University of Auckland
szhe024@aucklanduni.ac.nz, ian@cs.auckland.ac.nz

Abstract. *Real-Time Strategy (RTS) games have become an attractive domain for AI research in recent years, due to their dynamic, multi-agent and multi-objective environments. Micromanagement, a core component of many RTS games, involves the control of multiple agents to accomplish goals that require fast, real time assessment and reaction. In this paper, we present the application and evaluation of a Neuroevolution technique in evolving micromanagement agents for the RTS game Starcraft: Brood War (SC:BW). The NeuroEvolution of Augmented Topologies (NEAT) algorithm, both in its standard form and its real-time variant (rtNEAT) is comparatively evaluated in micromanagement tasks. Preliminary results suggest the general viability of these techniques in comparison to traditional, non-adaptive AI. Further analysis of each algorithm identified differences in task performance and learning rate.*

Keywords: Real-Time Strategy Games, Neuroevolution, Evolutionary Computation.

1 Introduction

It was predicted more than a decade ago, that interactive computer games would emerge as an ideal platform for Artificial Intelligence research [1]. Due to their increasingly complex and realistic simulations, video games have become fine approximations of real world environments. AI techniques can be developed and evaluated in a cost effective and contained manner, before being applied to more complicated real world problems [1,2]. The popularity of video games as an entertainment medium has resulted in a consistently growing, multi-billion dollar software industry [3]. This in turn is a driver of video game technology and research, of which AI is a vital component [4].

The popularity and ease of access to videogame hardware and software has increased the accessibility of computing power and simulation environments for AI research. On the other hand, the contribution of AI research to commercial game development has been lacking in recent years [5]. This has resulted in high dependency on deterministic and non-adaptive AI techniques in commercial games that limit their realism, replayability and challenge [6].

Real Time Strategy (RTS) games are a genre of video games that provide unique challenges to AI research [7]. Characteristic of the genre is a real-time,

S. Cranefield and A. Nayak (Eds.): AI 2013, LNAI 8272, pp. 259–270, 2013.

stochastic environment, with multiple objectives and enormous action and state space. These features require AI agents with multiple levels of abstraction and reasoning, fast reaction and expert game knowledge. An example of the genre is Starcraft: BroodWar (SC:BW)[1], an RTS game that is a popular game environment for AI research. Using a third party plugin called the Brood War API (BWAPI)[2], it is possible to create complex AI agents to play matches of SC:BW.

In this paper, we aim to evaluate the effectiveness of Neuroevolution (NE) techniques in developing learning agents for playing SC:BW. The goal is to contribute to the development of a complete AI system capable of learning and executing human expert level strategy in SC:BW. In particular we focus on 'micromanagement', a crucial level of abstraction in the RTS domain, handling the fast combat component of the overall game. NE applies evolutionary algorithms to train artificial neural networks that are known to be effective approximators of complex, non-linear functions. Meanwhile, RTS games have a large state and action space that is suitable for neural networks. Furthermore, research on the NEAT algorithm [8] has shown the effectiveness of NE in reinforcement learning tasks, of which SC:BW has been successfully modelled [9,10].

We first implemented a micromanagement agent for SC:BW that uses NEAT and a real time variant rtNEAT for learning behavior. Next, the viability of the agent was evaluated against the existing SC:BW AI in multiple experiments. Finally, we analyzed the difference in performance between standard NEAT and the real-time variant, both in the rate of learning and in match performance. The rest of the paper is structured as follows: we provide a survey of related work around SC:BW AI and the NEAT algorithm. Next, we describe an overview of the implementation of our AI agent and the usage of NEAT, followed by the evaluation of the agent and a discussion of results. Finally, we give concluding remarks and highlight areas of future research.

2 Related Work

In many RTS games, the game strategy can be roughly divided into two levels of abstraction. Macromanagement is the level that is concerned with high level strategic decision making such as resource planning and opponent modeling. Techniques dealing with macromanagement must choose and adapt sequences of strategic actions to meet goals of varied hierarchy. Example of techniques applied to this domain include Case Based Reasoning[11] and Goal Driven Autonomy[12]. Micromanagement is the level concerned with direct combat tactics and unit control. Traditionally, micromanagement is accomplished via static AI techniques such as scripts based on simple metrics [13]. More complicated techniques in the literature include Reinforcement Learning (RL) and Evolutionary Algorithm approaches.

RL combined with neural networks has been applied to SC:BW micromanagement [10]. Agent learning was accomplished using the online Sarsa RL algorithm,

[1] Starcraft: Brood War: `http://us.blizzard.com/en-us/games/sc/`
[2] Brood War API: `http://code.google.com/p/bwapi/`

with neural-networks to approximate the state-action value function. Results showed a significant winning advantage against standard Starcraft AI, but required thousands of training rounds and are limited in the type and number of units represented. In [9], a comparative evaluation of RL techniques applied to SC:BW was presented. Four variants of RL algorithms were applied to a specific micromanagement task, involving a long ranged unit with high mobility against numerous melee (close ranged) enemies. Evaluations identified strengths and weaknesses of the different algorithms, and showed a high win rate against the default SC:BW AI. However, the results are derived from a very limited scenario, and the author acknowledges it is only the first part of a larger RL based SC:BW agent.

Work that is most related to ours is from [14], in which rtNEAT was applied to SC:BW micromanagement. Units were controlled by separate neural networks, specifying actions to take in real time. The network typology is evolved over generations of 12 vs 12 unit combat. Evaluations showed a significant win rate within 300 training generations and also claimed the rtNEAT algorithm allowed fast, real-time strategy adaptation. A limitation of the study is the use of a custom SC:BW map that replenishes unit numbers with up to 100 unit reserves. This is an unrealistic depiction of real SC:BW combat where units are not replenished immediately to replace dead units. Furthermore, real-time fitness improvement occurs only over unit combat time that is minimal in a full SC:BW match. However, the work showcased the potential of applying NEAT based algorithms to SC:BW micromanagement that our work analyzes further.

2.1 Neuroevolution and NEAT

Neuroevolution (NE) has shown effectiveness compared to standard RL, in problems with continuous and high-dimensional state spaces [8]. Traditional NE worked on pre-defined topologies, and searched over the space of connection weights. Topology and Weight Evolving Artificial Neural Networks (TWEANNs) attempts to also evolve the topology of the network, and has the potential to improve training speed and accuracy of solutions [15]. Furthermore, it reduces the uncertainty and effort of deciding on network topology by researchers [16].

However, TWEANN techniques face numerous challenges, such as complications with network structure in crossover and problems with genetic encoding. Work by [16] developed the NEAT algorithm to address these challenges. The classic NEAT algorithm was expanded to a real-time variant[17], in which evolution occured over a real time environment. The rtNEAT algorithm was demonstrated in a game called NERO, where agents are evolved and adapted in real time to tackle changing objectives. Regular NEAT has been successfully applied to RTS by [18], where neural networks are evolved to become AI players in an ensemble process. In [6] both NEAT and rtNEAT were used to automatically balance the challenge of the AI player in an RTS game. A novel challenge metric coupled with a fitness function guided the evolution of neural networks. The AI was continuously evolved to converge to the same challenge level as the human player, thus creating a more balanced gaming experience.

3 Implementation

We use the BWAPI open source framework for creating and executing AI modules. It exposes functionality to retrieve information about the game state and to issue commands to game units. Units are encapsulated as BWNEAT units, with an accompanying neural network for decision making. The SC:BW game state is updated once per frame, i.e. every 56 milliseconds on normal game speed. BWAPI triggers an event on every game state update, allowing AI code to react. During this event, each BWNEAT unit feeds internal and external percepts from the AI Module as inputs to its neural network, and interprets the network output as the next action to be performed. The NEAT Manager module is responsible for instantiating the BWNEAT Units and is an interface to the NEAT and rtNEAT algorithms. It receives evaluated fitness from each unit, performs NEAT evolution and reassigns neural networks to BWNEAT units. Fig. 1 summarizes the agent architecture.

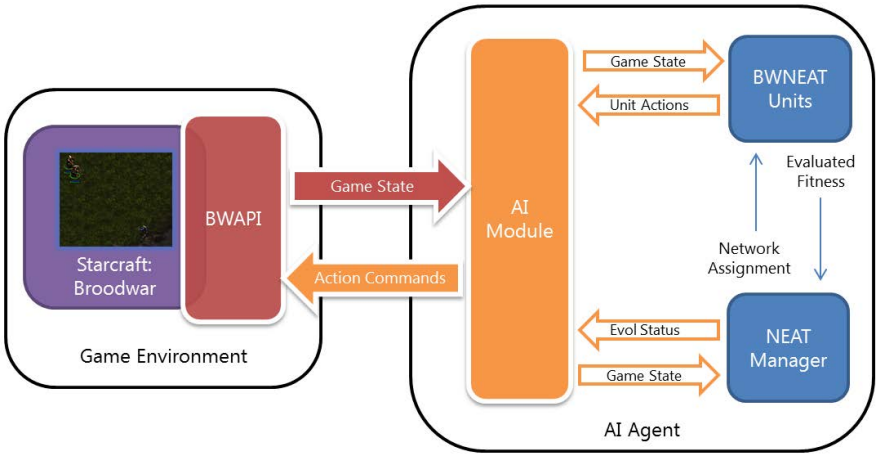

Fig. 1. An overview of the agent architecture

3.1 Neural Network Architecture

The basic neural network architecture is fully connected and feed forward, with randomized starting weights (Fig. 2). It begins with 0 hidden nodes and gradually allows nodes and connections to be added via the NEAT and rtNEAT algorithms. The inputs were chosen as important percepts to induce learning behavior that would allow a unit to inflict as much damage to the enemy units, while taking as little damage as possible. For example, when the unit's weapon is on cooldown (a pause between consecutive attacks), it should do its best to avoid damage. Another aspect is the range of unit weapons. If for example, the unit has a longer weapon range than the enemy units, it is possible to perform a hit-and-run strategy. These percepts are based on domain knowledge of SC:BW and is common in many RTS games.

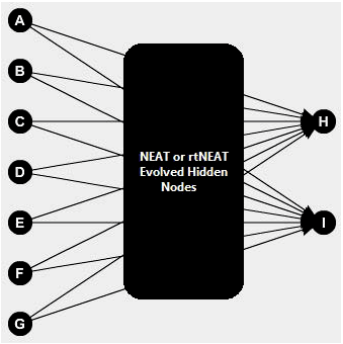

Fig. 2. Initial network architecture. Nodes A to G (*Bias, WeaponCooldown, RemainingHealth, WeaponRange, EnemyWeaponRange, NumAlliesInRange and NumEnemiesInRange*) denote a mixture of agent internal and external percepts as input to the network, while nodes H and I (*Fight and Retreat*) denote the outputs as two possible unit actions.

The output of the neural network corresponds to two unit actions: fight or retreat. The action with the largest corresponding output is chosen by the unit. If the fight action is taken, the unit executes a simple routine that targets the enemy unit with the lowest hit points (health) within its weapon range. The retreat action makes the unit move a small distance away from enemies and obstacles, via a weighted vector. These actions are based on similar implementations in [9] and [10], as they are simple to implement, but complicated enough to produce sophisticated behavior when performed in varying sequences.

3.2 Fitness Function

The NEAT and rtNEAT algorithms are guided by a fitness metric. In the context of SC:BW unit micromanagement, the fitness should reflect the performance of an individual unit. For both NEAT and RTNEAT, we define the fitness F_i for a unit $_i$ as:

$$F_i = \frac{TDD_i - HPL_i}{IHP_i} + 1 \tag{1}$$

The function takes in the total damage dealt by the unit (TDD), its hit point loss (HPL) accrued over the match and its initial hit point (IHP). In theory, the fitness is only upperbound by the total hit points of enemy units. However in practise, the average fitness of each unit falls under $[0, 2]$ where at its lowest the unit has produced no damage and dies, and at its highest value it has dealt twice as much damage than it has taken.

3.3 NEAT Evolution

Training via the classic NEAT algorithm occurs over generations of SC:BW matches. After a match, regardless of win or loss, the population of neural

networks go through an evolutionary process. First the fitness of each network is evaluated based on the units performance during the match. Next, some of the worst performing networks are replaced by the offspring of some of the best performing networks. This simple process is guided by three principles: tracking evolution via historical markers, protecting innovation via speciation and minimizing search via 'complexification' [8].

NEAT uses historical markings to efficiently evaluate similarity between network topologies. Networks are then speciated using a similarity metric formed via the number of disjoint genes D (genes that exist in one network and not the other), excess genes E (genes that appear in one network later in evolution than any genes on the other network) and the mean weight difference of matching genes W[8]:

$$S = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 W \tag{2}$$

$c1$, $c2$ and $c3$ are adjustible weighting coefficients, and E and D are normalized by dividing N, the number of genes in the larger network. Networks are grouped into species via this similarity metric, and a compatibility threshold that can be modified to specify species bounds. A network shares its evaluated fitness with other members of its species, in order to encourage diversification of solutions and prevent single species dominance. NEAT adjusts each network's fitness based on its similarity metric against all other organisms in the population. The number of offsprings spawned by a species after each generation is based on the proportion of its average species fitness to the total of all average species fitness [8].

3.4 rtNEAT Evolution

The real-time variant of the NEAT algorithm is designed specifically to operate in a continuous, real-time domain. In particular when adapted to video games, the performance of AI agents is able to improve gradually as the game is played, without abrupt changes over a whole generation of evolution. In the context of SC:BW, rtNEAT applies evaluation and replacement on game units every n ticks of game time. The number of game ticks between replacement is an important factor that affects evolution. If new organisms are replaced too quickly, then they cannot be evaluated accurately and new innovations may be needlessly thrown away. A law of eligibility is formed by [17], stating the number of ticks between replacements, with respect to the fraction of the population that is too young to be replaced I, the minimum time alive m and the population size P:

$$n = \frac{m}{|P|I} \tag{3}$$

In our experiments we empirically define m as 300 game frames, to offset the delay between the start of a match and the first enemy encounter. We follow [17] in defining 50% of the population as eligible for replacement, and $p = 12$ the number of units which is constant. This gives us $n = 50$, the number of

games frames between replacement in rtNEAT experiments. In the next section we describe in more detail, evaluations that incorporate these algorithms and principles to analyse the effectiveness of NEAT and rtNEAT for SC:BW micromanagement.

4 Experimentation and Results

We devised experiments to gauge the effectiveness of NEAT and rtNEAT evolved micromanagement agents against the standard SC:BW AI. A variety of unit setups were used in order to simulate different micromanagement scenarios. From these experiments, we saw very high fluctuation and variation in the fitness and win rate of agents over generations. In order to adjust for these fluctuations, we ran experiments to find the number of generations taken for each algorithm to converge to a suitable solution, when evolution is halted upon finding a potential candidate.

4.1 Experiment Setup

SC:BW units vary on attributes such as race, weapon and armour type. In order to keep the experimental variables constant and to avoid an explosion of unit type permutations, we based our experimental setup on [14] that compared 4 unit type variations (melee vs. melee, ranged vs. ranged, melee vs. ranged, ranged vs. melee). The number of units is kept at a constant 12 vs. 12, which is the maximum selectable number of units for a human controlled squad. The scenario used throughout experimentation is a flat map, based on those used in the AIIDE 2010 Starcraft micromanagement tournament[3].

4.2 Evolutionary Process Experiment

We first compared the performance of NEAT and rtNEAT algorithms on each of the 4 unit matchup variations, over 300 generations of evolution. Each matchup is repeated 25 times to reduce randomness in network starting weights. The average unit fitness and the match outcome is recorded over 300 generations, and averaged over the 25 runs.

Match Variations	NEAT WR	NEAT SD	NEAT CL 95%	rtNEAT WR	rtNEAT SD	rtNEAT CL 95%
Range vs. Range	60.39%	9.86%	1.12%	72.35%	11.53%	1.31%
Range vs. Melee	97.59%	7.95%	0.90%	69.48%	10.44%	1.19%
Melee vs. Melee	23.80%	8.26%	0.94%	49.73%	11.63%	1.32%
Melee vs. Range	58.89%	10.79%	1.23%	47.87%	10.26%	1.17%

Fig. 3. Summary statistics for our first experiment. Mean win rate (WR) over 300 generations, standard deviation (SD) and the 95% confidence level (CL) is shown.

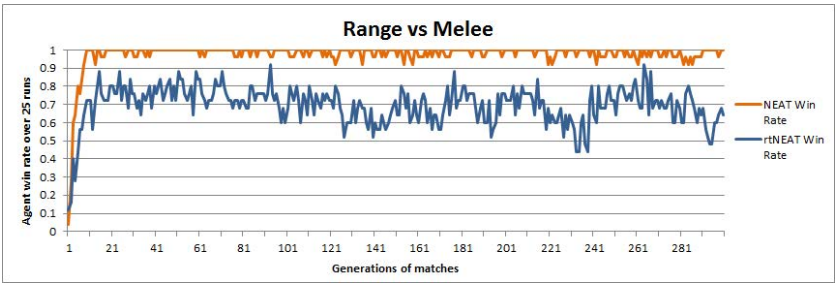

Fig. 4. Plot of the best performing match up (range vs. melee) average win rate

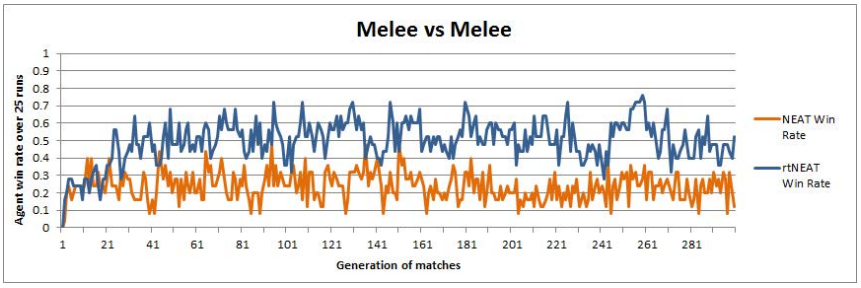

Fig. 5. Plot of the worst performing match up (melee v.s melee) average win rate

The results suggest that there is no single algorithm dominating all match variations (Fig. 3). Mean win rate is higher for NEAT on range vs. melee (mean 97.59%, SD 7.95%) and melee vs. range (58.89% mean, 10.79% SD), while rtNEAT is higher on range vs. range (60.39% mean, 9.86% SD) and melee vs melee (49.73% mean, 11.63% SD). If we consider a win rate higher than 50% to indicate better than baseline performance against the built-in SC:BW AI, then both NEAT and rtNEAT show effectiveness on range vs. range and range vs. melee battles. NEAT is also effective in melee vs. range (58.89% mean $\pm$ 1.23% at 95% confidence level). From examining the agent behavior, the effectiveness of controlling ranged units can be attributed to having learnt the hit-and-run micromanagement strategies. It is interesting to note the generally poor performance of both algorithms when controlling melee units. We discuss in more detail the evolved behavior of the agents contributing to the performance in Section 5.

Fig. 4 and Fig. 5 show plots of some of the best and worst performing match up variations. The average fitness plot is not shown, but is highly correlated to the average win rate. From these plots, we see a trend of initial poor performance and a quick convergence to some local optima. This is typical of evolutionary algorithms where the initial starting solutions are randomized and are

[3] AIIDE 2010 micromanagement tournament: `eis.ucsc.edu/starcrafttournament1`

not expected to perform well. On all variations the first convergence to a local optimal occurs between the 10th to 20th generation. There is significant fluctuation of win rate throughout generations that is further illustrated by high variance shown by their standard deviations. In general, the standard deviation is higher on rtNEAT runs, suggesting greater variation in evolutionary success over generations than standard NEAT. This may be due to the nature of the rtNEAT algorithm, in introducing real time change. Both algorithms are capable of producing high performing solutions at various generations, but are also quick to introduce mutations weakening the solutions. This is partly due to the nature of the experiment where we allow evolution to continue even after achieving a winning solution.

4.3 Generational Convergence Experiment

By defining a success criteria, we can halt the evolution of both the NEAT and rtNEAT algorithms once an acceptable solution is achieved. We defined a successful solution to be an agent that achieves 10 consecutive wins (the probability an agent with 50% win rate can win 10 consecutive games is $< 0.1\%$). This is a strict criteria as an agent may still be high performing even though it loses 1 game out of 10 (e.g. due to the stochastic nature of the game state). However, we use this to simplify the running of the experiment and to show that it is possible to robustly generate agents of this level of performance. We keep all other variables the same as in our previous experiment, except that the evolution terminates when a solution reaches 10 wins, or after 1000 generations. When a solution achieves a win, evolution is halted until the agent either achieves 10 wins, or a loss is encountered, where upon evolution continues. After a successful solution is achieved, or if no solution is found after 1000 generations, the experiment is reset to an initial population with randomized weights. For each algorithm and each matchup, we stopped the experiment at 60 runs and analyzed the results.

Match Variations	NEAT MNG	NEAT SD	NEAT CL 95%	rtNEAT MNG	rtNEAT SD	rtNEAT CL 95%
Range vs. Range	116.03	124.39	32.13	18.33	12.52	3.24
Range vs. Melee	4.15	1.76	0.46	19.95	26.03	6.73
Melee vs. Melee	42.52	53.15	13.73	26.78	18.30	4.73
Melee vs. Range	9.60	5.90	1.52	22.07	16.00	4.13

Fig. 6. Summary statistics for generational convergence experiment. Mean number of generations (MNG) taken to produce an acceptable solution is shown. Standard deviation (SD) and the 95% confidence level (CL) for the mean was also calculated.

In all experiments, an acceptable solution was found before 1000 generations, with most converging under 100 generations (Fig. 6). There was high variability in the number of generations required to arrive at an acceptable solution, evident by the high standard deviation in some match ups (e.g. 124.39 SD and 116.03 Mean generations for NEAT range vs range). The mean number of generations

taken between NEAT and rtNEAT is comparable to the average win rate performance of the previous experiment: NEAT converges faster for range vs. melee and melee vs. range match ups, while rtNEAT is faster for range vs. range and melee vs. melee. The range in generations taken between different matchups is higher for NEAT (4.15 mean for range.vs melee and 116.03 mean for range vs. range) than for rtNEAT (18.33 mean for range vs. range and 26.78 mean for melee vs. melee). This suggests the performance of rtNEAT is more stable under different unit variations.

Overall, the experiment showed that both algorithms were capable of generating effective solutions for micromanagement against the default SC:BW AI. However, it was necessary to establish an acceptance criteria for which to halt evolution and to preserve winning behavior. In the next section, we discuss further implications of the experimental results.

5 Discussion

In the first experiment, the fluctuation of fitness and success rate of solutions can be due to a number of reasons. Firstly, it suggests that any structural innovations introduced were making significant differences in the performance of the neural networks. This is probably due to the simplicity of the network design, where only 2 outputs exist, such that any structural change may affect the action selected. A simple neural network allows faster convergence by reducing the search space of initial nodes and weights. But it also means it is faster to diverge from the local optima. On top of this, the stochastic nature of the game environment can result in the same solution having varied success over different runs.

This also explains the general poor performance of both algorithms on melee match ups in the first experiment. Melee units do best in direct attack as they lack the weapon range to perform hit-and-run maneuvers. Any innovation introduced to make melee units run will immediately reduce the success rate of the solution. In the second experiment, the algorithms have no problem generating a solution for melee match ups, when no new innovations were introduced after a solution begins to do well. Another factor is an interesting behavior exhibited by the units over generations of evolution: some units are evolved to retreat when enemies are first found, but come back to fight after allied units are engaged in combat. These units tend to generate more fitness than those directly attacking from the beginning, as they do not receive as much damage over time. However, as the population begins to favour this behavior, there is a breaking point in which no units will stay to fight, leading to a match loss and a return to evolution favouring units that do not retreat. This cycle is highly correlated with the fluctuation of the win rate over generations.

Interestingly, the first experiment showed that rtNEAT produced higher variation in the success of solutions than NEAT over time. However in the second experiment, the average number of generations for an acceptable solution was less varied across different match ups than NEAT. This suggests real-time evolution can be quicker in introducing changes that reduce fitness, but also allow a

more robust convergence to a solution regardless of unit variation (variability in state and solution space). This is intuitive, since rtNEAT should be faster in reacting to changes in the environment in real time, than regular NEAT evolution between generations.

It is possible to complicate the initial neural network architecture, by incorporating more percepts as input nodes and providing finer grain output decisions. For example, the inputs can incorporate a deeper ontology of unit quality and type variations (armour, weapon and ability types etc) and more precise directional and distance data. Instead of fight or retreat actions, the decisions can be to move at specific angles for specific distances, and to explicitly decide which units to attack. Enemy target selection is itself complicated enough to be a separate learning task, perhaps requiring the optimization of a separate neural network that takes into consideration enormous unit type variations and the location of units. More complicated neural network designs allow for agents with more complicated behaviors that are able to perform well under a higher variety of conditions. The disadvantage is a greater number of dimensions to search and optimize for, resulting in slower training time.

There are limitations to the evaluation methodology to be addressed. For example, while the experiments show that the technique is able to learn to defeat the standard SC AI, the results do not extend to human opponents. However, testing against the standard SC AI is a baseline measure used in much of the related work, particularly for micromanagement. It is difficult to evaluate against human players, due to the number of games required to be played, and the lack of an objective human skill measure for the micromanagement task (current measures exist only for full SC games). It is possible to evaluate against other micromanagement AI, but there is a lack of a standardized evaluation methodology to do so.

6 Conclusions

Our evaluations confirmed the viability of NEAT and rtNEAT algorithms in evolving agents for various SC:BW micromanagement scenarios. When the algorithms are allowed to run non-stop, the win rate of agents against the default SC:BW AI fluctuates highly over generations. However, when evolution is halted upon reaching an acceptable level of performance, both algorithms are able to consistently generate winning agents, with most under 100 generations. Each algorithm differs in the variability of performance over different unit matchups. Factors contributing to the difference in performance include the complexity of the network starting topology and the variation in unit types. There is room to explore network complexity further, and a need to establish standardized evaluation methods for micromanagement agent evaluations. More work is needed to adapt these techniques for commercial RTS game deployment, but results here have shown promising performance in a learning AI capable of defeating scripted AI under short training time.

References

1. Laird, J., VanLent, M.: Human-level AI's killer application: Interactive computer games. AI Magazine 22(2), 15–26 (2001)
2. Buro, M.: Call for AI research in RTS games. In: Proceedings of the AAAI 2004 Workshop on Challenges in Game AI, pp. 2–4 (2004)
3. Siwek, S.E.: Video Games in the 21st Century. Technical report. Entertainment Software Association (2010)
4. Yildirim, S., Stene, S.B.: A survey on the need and use of ai in game agents. In: Proceedings of the 2008 Spring Simulation Multiconference, pp. 124–131 (2008)
5. Mehta, M., Ontañón, S., Amundsen, T., Ram, A.: Authoring behaviors for games using learning from demonstration. In: Workshop on Case-Based Reasoning for Computer Games, ICCBR (2009)
6. Olesen, J.K., Yannakakis, G.N., Hallam, J.: Real-time challenge balance in an RTS game using rtNEAT. In: 2008 IEEE Symposium on Computational Intelligence and Games, pp. 87–94 (2008)
7. Buro, M., Furtak, T.M.: RTS games and real-time AI research. In: Proceedings of the Behavior Representation in Modeling and Simulation Conference, pp. 63–70 (2004)
8. Stanley, K.O., Miikkulainen, R.: Efficient Evolution of Neural Network Topologies. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002). IEEE (2002)
9. 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), pp. 402–408 (2012)
10. Shantia, A., Begue, E., Wiering, M.: Connectionist reinforcement learning for intelligent unit micro management in starcraft. In: The 2011 International Joint Conference on Neural Networks (IJCNN), pp. 1794–1801 (2011)
11. Cadena, P., Garrido, L.: Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in StarCraft. In: Batyrshin, I., Sidorov, G. (eds.) MICAI 2011, Part I. LNCS, vol. 7094, pp. 113–124. Springer, Heidelberg (2011)
12. Weber, B., Mateas, M., Jhala, A.: Applying goal-driven autonomy to StarCraft. In: Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010 (2010)
13. Davis, I.L.: Strategies for strategy game AI. In: Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Computer Games, pp. 24–27 (1999)
14. Gabriel, I., Negru, V., Zaharie, D.: Neuroevolution based multi-agent system for micromanagement in real-time strategy games. In: Proceedings of the Fifth Balkan Conference in Informatics - BCI 2012, p. 32 (2012)
15. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE 87, 1423–1447 (1999)
16. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. 10(2), 99–127 (2002)
17. Stanley, K.O.: Evolving neural network agents in the NERO video game. In: Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games, pp. 182–189 (2005)
18. Jang, S.H., Yoon, J.W., Cho, S.B.: Optimal strategy selection of non-player character on real time strategy game using a speciated evolutionary algorithm. In: Proceedings of the 5th International Conference on Computational Intelligence and Games, pp. 75–79 (2009)