Author: J. M. Traish and J. R. Tulip

Title: Towards Adaptive Online RTS AI with NEAT

Year: 2012

Editor: C. proceedings

Conference Name: IEEE Symposium on Computational Intelligence and Games (CIG)

Conference Location: United States

Publisher: IEEE

Date: 11-14 September 2012

Abstract: Real Time Strategy (RTS) games are interesting from an Artificial Intelligence (AI) point of view because they involve a huge range of decision making from local tactical decisions to broad strategic considerations, all of which occur on a densely populated and fiercely contested map. However, most RTS AI used in commercial RTS games are predictable and can be exploited by expert players.Adaptive or evolutionary AI techniques offer the potential to create challenging AI opponents. Neural Evolution of Augmenting Technologies (NEAT) is a hybrid approach that applies Genetic Algorithm (GA) techniques to increase the efficiency of learning neural nets. This work presents an application of NEAT to RTS AI. It does so through a set of experiments in a realistic RTS environment. The results of the experiments show that NEAT can produce satisfactory RTS agents, and can also create agents capable of displaying complex in-game adaptive behavior. The results are significant because they show that NEAT can be used to evolve sophisticated RTS AI opponents without significant designer input or expertise, and without extensive databases of existing games.

DOI: http://dx.doi.org/10.1109/CIG.2012.6374187

URL: http://researchoutput.csu.edu.au/R/-?func=dbin-jump-full&object_id=41227&local_base=GEN01-CSU01

Author Address: JTraish@csu.edu.au/ jtulip@csu.edu.au/

CRO identification number: 41227

Towards Adaptive Online RTS AI with NEAT

Jason M. Traish and James R. Tulip, *Member, IEEE*

***Abstract*—Real Time Strategy (RTS) games are interesting from an Artificial Intelligence (AI) point of view because they involve a huge range of decision making from local tactical decisions to broad strategic considerations, all of which occur on a densely populated and fiercely contested map. However, most RTS AI used in commercial RTS games are predictable and can be exploited by expert players.**

Adaptive or evolutionary AI techniques offer the potential to create challenging AI opponents. Neural Evolution of Augmenting Technologies (NEAT) is a hybrid approach that applies Genetic Algorithm (GA) techniques to increase the efficiency of learning neural nets. This work presents an application of NEAT to RTS AI. It does so through a set of experiments in a realistic RTS environment.

The results of the experiments show that NEAT can produce satisfactory RTS agents, and can also create agents capable of displaying complex in-game adaptive behavior. The results are significant because they show that NEAT can be used to evolve sophisticated RTS AI opponents without significant designer input or expertise, and without extensive databases of existing games.

I. Introduction

Video games have advanced in graphics and physics to the point of diminishing returns. However, video game artificial intelligence (AI) has not advanced to the same extent. As a result there has been a surge of interest in video game AI as a differentiating factor in commercial games. This has been reinforced by the advent of multicore processors in the consumer market, the power of which current games find difficult to exploit.

There are particular challenges for developing RTS game AI. These include managing resource collection, tactics, research, reconnaissance, and high level strategy in a partially observable environment. A central issue for all RTS AI techniques is the extremely large state space that must be searched for solutions, due to the complex nature of RTS games. The intrinsic complexity of RTS games has attracted research interest which previously focused on board game AI such as for chess or go.

J. M Traish is with the School of Computing and Mathematics, Charles Sturt University, Bathurst, NSW, Australia. (phone: 61-2-63384560; e-mail: jtraish@csu.edu.au).

J. R Tulip is also with the School of Computing and Mathematics, Charles Sturt University, Bathurst, NSW, Australia. (e-mail: jtulip@csu.edu.au).

AI techniques for Real Time Strategy (RTS) games, such as Finite State Machines (FSMs) and scripting, produce predictable behaviors which can be exploited by expert human players. Expert players are forced to seek challenge in an online environment where their skills are pushed to their limit by other players

McCoy and Mateas [1] demonstrated an 'integrated agent' capable of playing an entire game well. The agent was made up of a system of individual managers, each responsible for a different aspect of the game. However, their agent was based on a deep understanding of successful RTS play, and extensive expert experience in many games.

One avenue of research which has potential application to developing challenging new RTS AI is adaptive or evolutionary AI. Such AI can learn, leading to more competitive AI behaviors. Adaptive AI could also automate much of the fine tuning of AI agents for RTS games. The video game industry is cautious in using adaptive techniques, but adaptive techniques do offer the promise of generating more challenging AI opponents without vastly increasing development costs.

Some adaptive techniques applied to RTS AI include Dynamic Scripting (DS), Case-Based Reasoning (CBR), and Artificial Neural Nets (ANN). Examples include [2] who used Reinforcement Learning (RL) to optimize the time taken to build a predefined number of buildings, [3]-[4] who trained an ANN to place defensive structures in a constrained game environment, and [5] who used Case Based Reasoning (CBR) to predict player behavior. Ponsen [6] developed an adaptive system based on DS. His AI operated at the strategic decision level. The agent made decisions in relation to how many workers to create for resource gathering, and specified the size and type of attacking force, while the game environment handled path-finding and the details of tactical combat.

However, CBR relies on extensive databases of played games and is susceptible to quality issues in those games. ANNs also require extensive training sets of played games, and have issues with learning speed due to the high dimensional decision space involved in RTS games. DS is effective in evolving counters to specific opponents, but requires expert knowledge to develop rules that cope with varying opponent strategies.

Neuro-Evolution of Augmenting Topologies (NEAT) is a

hybrid genetic and neural net technique developed by Ken Stanley in [7] that addresses issues of training speed and training set size faced by conventional ANNs. It can also operate effectively in high dimensional decision spaces. It has been successfully applied as a game AI in other genres such as First Person Shooters (FPS) [8].

NEAT has been applied to evolve single strategy RTS agents and to dynamically adjust difficulty levels [9]. However, this work examines the application of NEAT as an adaptive RTS AI at the strategic decision level. It focuses on high level strategy, involving economic, military production, attack and defense types of decision. Unit level micromanagement such as formation management, squad management, and combat tactics are not dealt with. The high level decision model is similar to Ponsen's approach in [6].

This is of interest because NEAT has not been applied to RTS AI before and offers the potential of evolving complex behavior. If successful the technique could help reduce the workload of developing challenging RTS AI, and also help in the balancing and tuning of RTS games.

The paper is organized in several sections. Section 1 sets the context, and identifies the aim of the research. Section 2 develops the experimental approach applied in this work. Section 3 presents the results of the experiments. Section 4 discusses the results in the context of existing work; it then discusses the research questions arising from the work and potential avenues for further work. Section 5 draws some final conclusions.

II. Experimental Approach

The research questions addressed in this work are;

1. Can NEAT generate an effective AI agent for RTS games?
2. Can NEAT create agents with complex and varying behavior when there is no single successful strategy?

These questions have been addressed by evaluating the performance of NEAT in a series of experiments.

In Experiment 1, NEAT based agents were evolved against a range of single static AI opponents. The intent of this experiment was to see if NEAT could evolve effective RTS AI agents, and to establish the basic mechanisms for doing so. A range of standard static AI strategies were trialed to provide comparisons with the established body of work in the area.

In Experiment 2, agents were evolved against the same range of static AI as in Experiment 1. However, in Experiment 2 the agents were required to defeat all of the static AI. The aim was to see if agents with complex behavior would evolve. However, the results were not as expected.

The experiment was redesigned and a third experiment was run. In Experiment 3, a 'dominance tournament' system [10] was used. In a 'dominance tournament' the adaptive AI 'co-evolve' by adding successful challengers to a 'pool of champions', and requiring new challengers to defeat all champions before being added to the pool. Once again the intent was to see if NEAT would evolve complex behaviors in response to facing varying opposing strategies, but once again the results were not as expected.

A final experiment was designed in which a set of agents implementing a cycle of counters was forcibly evolved and then used as a set of static AI. In Experiment 4, once again the agents were given the task of defeating the whole set of opposing AI. This experiment was aimed at eliminating the possibility of a successful 'degenerate strategy' occurring, thus forcing complex behavior to evolve.

A. Experimental Setup

The experiments required an RTS environment that was fast, reliable, had a simple interface, had commonality with previous work, and in which the issues of unit balance had already been resolved.

The RTS environment chosen was Wargus. This is an open-source clone of the established commercial RTS game Warcraft 2, and has been used in [1] and [6]. It replicates a well-established RTS so that issues of unit definition and balance have been thoroughly resolved. It is a stable platform, and the interface for customized AI is well established. Finally, it was found to demonstrate very high simulation rates allowing for fast iterations. Although fairly old, Wargus was ideal for the purposes of this research.

B. Strategic Model

The strategic decision space is represented by a set of actions regarding building of economic and military production facilities, upgrading those facilities, training economic and military units of various capabilities, upgrading military units, and issuing orders to those units to attack. The model developed is referred to as the 'elective action' model. This model divides actions into elective or optional actions, and required or antecedent actions. Only elective actions can be selected, and the selection of an elective action causes all required antecedent actions to be carried out if necessary. This model allows a very simple input/output model to be utilized.

Fig 1 shows the relationships between elective and required actions used in the model. The diagram shows the dependencies of the in-game actions in the current experimental approach. Some actions can operate as either elective or required actions depending on the circumstances under which they are invoked.

Build actions consist of deciding what economic and military facilities to construct. Upgrade actions consist of upgrading those structures to produce advanced units such as ogres or catapults. Economic actions consist of deciding to train peons (worker units), how many peons to train, and what resources the peons are set to gather. Military production actions consist of training military units such as grunts, ogres, axe throwers, catapults, or dragons. Finally, attack actions consist of directing units to attack.

Attack actions specify the timing and frequency of attacks. The size of the attacking force is controlled indirectly through the interval between attacks. The attack

rules work in combination with military production rules to create a variety of force size, composition and attack frequency. Defensive actions were not required, since a non-attacking force is already in a defensive state and position.

agent to request multiple build orders in parallel, while not slowing building by depleting resources.

Game lengths were restricted to a maximum of 100,000 game cycles (equivalent to around 55 minutes of real-time

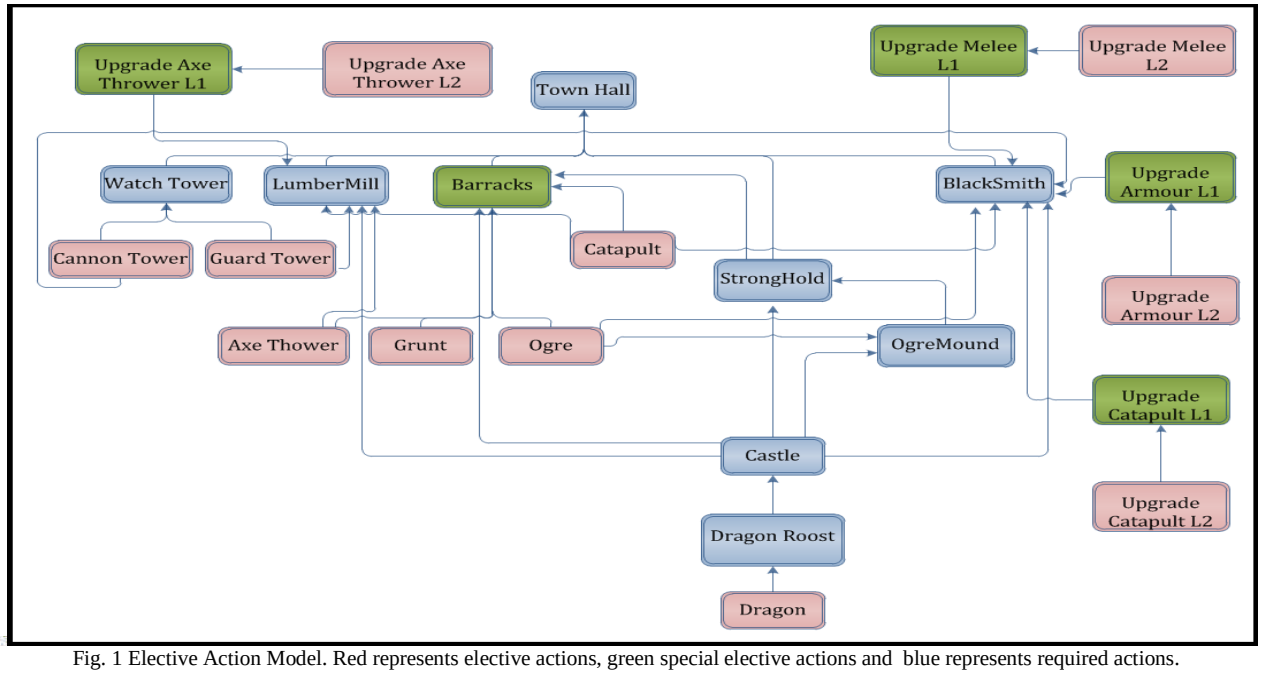

Fig. 1 Elective Action Model. Red represents elective actions, green special elective actions and blue represents required actions.

However defensive build actions such as constructing guard and cannon towers were included.

Several constraints on the maximum number of units and buildings were applied in order to ensure fast simulations. The constraints encouraged 'intelligent' use of differing units rather than relying on overwhelming force, and restricted agent decisions to some reasonably valid subset of all possible decisions.

C. NEAT Implementation

1) *Input Model:* Game objects such as units or buildings progress through four states in Wargus. To begin with they are 'requested'. When sufficient resources are available to fulfill the request, the game object moves to the 'queued' state, in which it is trained or constructed. Once completed, it becomes 'alive', and can receive commands. If it is destroyed, it moves to the 'dead' state.

The numbers of each type of game object for both the agent's, and the opponent are used as inputs for the NEAT AI. The sum of instances in both the queued and alive states for a particular type of game object was found to be the most useful representation for input to the ANN. If only 'alive' units were counted, the agent responded slowly, continuing to request units or buildings after their production in progress. If 'requested' objects were included, the agent would move on to make other requests, resulting in dissipation of effort.

Including only 'alive' and 'queued' game objects allowed the agent to focus on a particular type of game object until it was queued, but move on to other priorities once production of that unit or building was assured. This also allowed the

game play). If a game reached this limit, it timed out and the challenging agent was considered to have lost. This limit was imposed to avoid stalemated games which could severely hinder the evolutionary process.

2) Output Model: Output nodes are mapped to commands issued by the agent (see Fig 2). Only 'elective' or 'non-dependent' actions are used as output nodes, since an elective action includes all antecedent 'required' actions. Restricting output nodes to only 'elective' actions eliminated the request of invalid actions and reduced the solution state space complexity. A second restriction prevented build requests once a unit limit had been reached.

Output nodes were prioritized according to returned value, and only the highest priority action requested. The combination of the 'queued plus alive' input method and the 'prioritized request' output method resulted in an agent that was responsive to the environment, while remaining focused on high priority requests until resource availability allows them to be fulfilled.

Attack actions were handled slightly differently to other actions. Once the attack node was activated, attack orders were issued at intervals whose frequency depended on the value of the attack node. A high attack node value resulted in an immediate attack with subsequent continuous reinforcement. A low attack node value resulted in the building up of a large force, and a long interval between massive attacks, if the attack order ever became the highest priority.

2) Fitness Function: The fitness function was designed to result in aggressive agent behavior, and avoid the issue of

defensive bias and resulting stalemates.

The chosen fitness function assigned a challenger the maximum fitness value if it won, regardless of method, or how close a win occurred. For losing challengers an aggressive stance was rewarded, which is an essential behavior for winning and for avoiding long drawn out stalemates. Building an army larger than the opponent's was also rewarded, although the incentive was capped to avoid production an excessively large force. Although these reward criteria seem relatively specific, they are intended merely to guide selection towards the most basic of RTS strategies; build up an army and attack.

The influence on selection of the fitness calculated for a losing challenger was outweighed by the maximum fitness assigned to a winning challenger. This allowed various subtle and complex behaviors that lead to a victorious result to be rewarded and retained during the selection process.

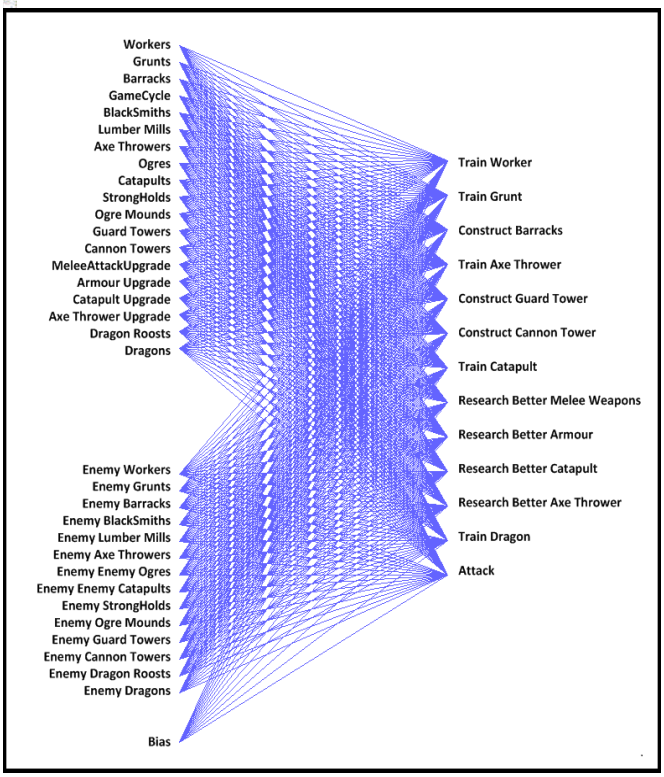

Fig. 2 Initial NEAT ANN.

D. *Wargus Environment*

The Wargus RTS environment was modified slightly to allow NEAT agents to interact with Wargus as an AI controller. Bias issues were avoided through creation of symmetrical, 'open path' maps, and requiring agents to play from alternate starting positions in a set of two games. Fitness was calculated as the average from both games.

2) Map Size: Map size has a large influence the number and diversity of successful strategies possible. Map sizes of 64*64 and 128*128 were chosen on the basis that these sizes allowed comparison with other established work.

3) Starting Configuration: Two starting configurations were chosen. In one, each player starts with a single worker (or 'peon'). This is the most common starting configuration for multiplayer games. In the other configuration, the player starts with both a 'Town Hall' and a 'Barracks'. This provides a shorter initial phase before interesting strategic decisions occur.

4) Static AI Opponents: Three differing static AI opponents used for the experiments. The strategies chosen were based on those used in previous work and were as follows;

1. Soldier Rush. This strategy trains a large force of cheap units and attacks early in the game.
2. Knight Rush. This strategy trains a small force of cheap units and attacks, then builds an ogre mound (ogres are advanced units, equivalent to knights), and trains a large force of advanced units, attacking when a large force is ready
3. Fast Ogre Rush. This strategy trains a large number of workers, builds an ogre mound as soon as possible, and then trains a large force of advanced units and attacks. The time taken to attack with the large force is much shorter than the time taken to attack in the Knight Rush because no defenses are built and there is no initial light attack.

III. Results

A. *Experiment 1: Single Static Opponent*

The intent of Experiment 1 was to see if NEAT could create agents capable of operating in an RTS environment. This question is non-trivial since the action or state space for an RTS is large, and it was not clear that any sort of reasonable behavior would result.

Fig 3 shows a plot of the average generation fitness over all repeat runs for the challenger in the 1 peon, small map configuration. In all configurations, NEAT was able to generate a successful challenger within 20 generations. The differences in run lengths and fitness values reflect the relative difficulty NEAT had with the opponents. It can be seen that in all cases NEAT had little trouble defeating the 'Fast Ogre' strategy, but generally found the 'Soldier Rush' more difficult.

On small maps, NEAT generally produced solutions favoring immediate attack, building only a single peon (worker unit) and attacking with grunts (the cheapest and most basic military unit) as soon as production facilities (barracks) were available. This behavior is a variant of the strategy known as the 'Soldier Rush'. Even against the static Soldier Rush AI, NEAT generated an optimized version of the Soldier Rush known as the 'immediate Soldier Rush' in response.

On large maps, NEAT produced a variety of strategies, but most conformed to a single overall 'meta-strategy'. This involved initially building a large number of guard towers, which could successfully defend against an early 'Soldier Rush'. Then, a variety of different units would be produced behind the defensive wall. These units ranged from grunts

through more advanced units such as axe-throwers, catapults, ogres, dragons, or any combination of these unit types. The 'Soldier Rush' AI's economy would suffer because of its emphasis on attack, and the NEAT challenger would overwhelm it with a late game attack. The strategies NEAT developed on large maps are known as 'turtling' in the RTS community. 'Turtling' refers to a strategy focusing on an initial strong defense, upgrading units where possible, building a large army of strong units, and attacking late in the game.

Experiment 1 demonstrated that NEAT could produce an effective 'integrated agent' RTS AI.

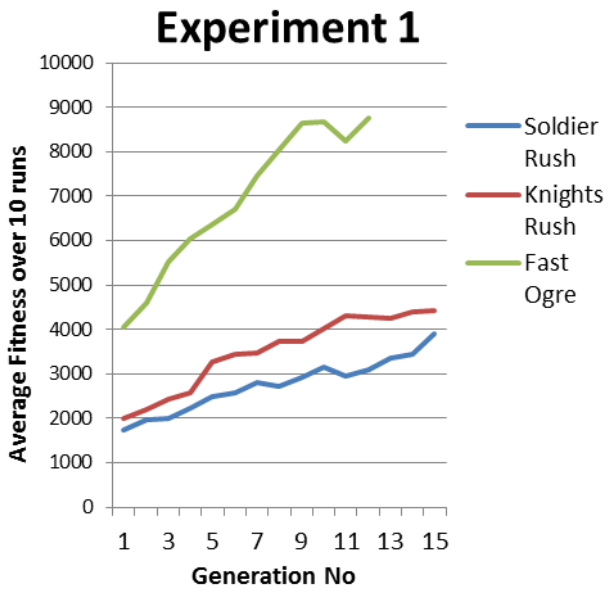

Fig. 3 Average NEAT challenger fitness in Experiment 1, 1 Peon, small map configuration.

B. *Experiment 2: Multiple Static Opponents*

The intent of Experiment 2 was to see if NEAT could evolve a single agent capable of defeating several different strategies by generating complex adaptive behavior.

In every case, NEAT was able to generate a successful challenger within 25 generations although the average population fitness was much lower than observed in experiment 1. However, in each run a single strategy agent evolved that could defeat all three opponent AIs. Fig 4 shows a plot of the average generation fitness over all runs for the NEAT challenger in experiment 2.

On small maps, once again NEAT produced a solution based on an immediate attack, while on large maps, the NEAT agent would adopt a turtling strategy, taking a defensive stand, building a large army, and then attacking late in the game. The lower diversity of solutions found on each map type is indicative of the much tighter solution constraints imposed by multiple opponents with differing strategies.

The fact that NEAT could find a single strategy to defeat the multiple opponents is strong evidence that the three static AI used in experiments 1 and 2 did not form a classic RTS cycle of counters. The result from experiment 2 simply showed that if there was a single 'degenerate strategy' solution to the multiple opponent problem, then a genetic technique could find it.

In order to see whether NEAT could evolve an agent with complex adaptive behavior, another experiment, which eliminated the possibility of a single solution, was required.

C. *Experiment 3: Dominance Tournament*

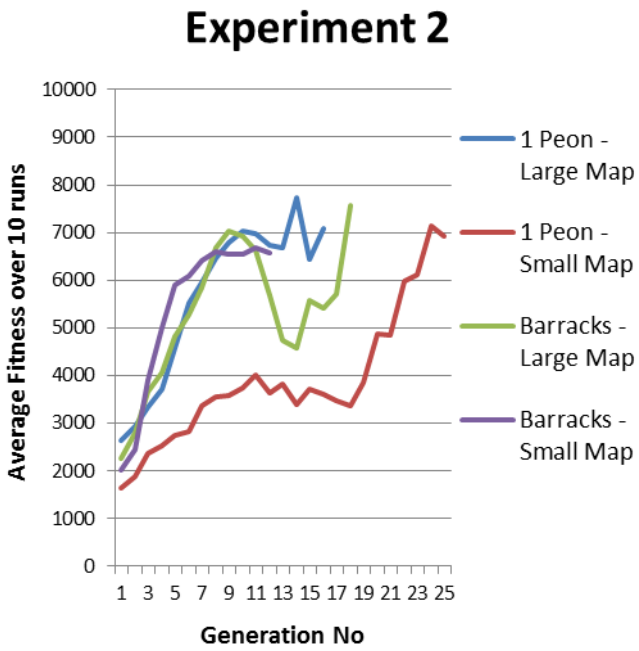

Fig. 4 Average NEAT challenger fitness in Experiment 2, multiple opponents, all configurations

In Experiment 3 a dominance tournament [16] was used. It was thought that either a single optimal solution would be found, or that eventually an agent would be forced to evolve complex behavior to cope with the differing champions.

However, game times were longer, and the number of champions grew much faster than expected, resulting in extremely long execution times for later challengers, and very slow evolution. This was because the dominance tournament produced a series of incrementally better versions of the initial 'Soldier Rush' strategy. This process repeated for the length of time the experiment was allowed to run. It may be that eventually, a 'leap' to a new strategy might occur, but this was not observed in the amount of time available for the experiments conducted.

Another experiment, which eliminated the possibility of a single solution and enforced the need for differing behaviors was required.

D. *Experiment 4: Cycle of Counters*

For Experiment 4, an initial, highly optimized 'immediate Soldier Rush' champion was created based on the results of Experiment 3. A challenger was then evolved to defeat the optimized champion as in Experiment 1. Finally a third

champion was evolved to beat the second champion, but with the additional requirement that it should lose to the first champion. In this way, a 'cycle of counters' was evolved. The final cycle was checked to make sure that no two champions represented refinements of a single strategy.

The strategies involved in the 'cycle of counters' were as follows:

1. *Champion 1: 'Optimized early game Soldier Rush'* This strategy builds 7 peons, and a barracks. It then prioritizes production of grunts and attacks immediately and continuously.
2. *Champion 2: 'Turtle with mid game Soldier Rush'* This strategy creates a quick defense of tower structures, a barracks, then builds up a large force of grunts, and attacks.
3. *Champion 3: 'Turtle with late game catapult rush'* This strategy initially builds ups its economy with many peons, and then builds many defensive structures. It then upgrades its military production buildings and produces a large number of catapults (advanced ranged units).

NEAT was able to generate a successful challenger for the 'cycle of counters' within 200 generations. Furthermore, the challenger exhibited differing strategies in response to each opponent AI. Experiment 4 finally demonstrated that NEAT could produce agents with complex adaptive behavior. Fig 5 shows a plot of the average generation fitness over all repeat runs for the NEAT challenger in experiment 4.

Against Champion 1 (optimized early game Soldier Rush) the agent built a barracks and produced grunts, but did not attack. Because no attack order was issued, an army of 5 or 6 grunts built up. As Champion 1's early attacking grunts entered the NEAT agents base, the agent's idle army of grunts easily defeated them, gradually becoming drawn towards Champion 1's base. When the NEAT army reached the opposing agent's base, they easily overwhelmed it, with reinforcements sent as an attack order was finally issued. Against Champion 2 (Turtle with mid game Soldier Rush), two distinct behaviors were observed in the alternate starting position games. In the positively biased position, the agent launched an early attack, similar to that of Champion 1. The attack succeeded before Champion 2 could erect its defensive line. In the negatively biased position, the agent once again launched an early attack, but was unable to succeed. The agent then stopped sending grunts into the failed attack, and built its economy, producing a large force of varied units but not attacking. When Champion 2 attacked with its large force of grunts, the agent had a large force of advanced units (including catapults) and destroyed the attacking force. The agent then attacked, eliminating Champion 2's defensive towers with its longer range catapults and then destroying Champion 2's base.

Against Champion 3 (Turtle with late game Catapult rush), the agent attacked early. This strategy was successful in both positively and negatively biased games.

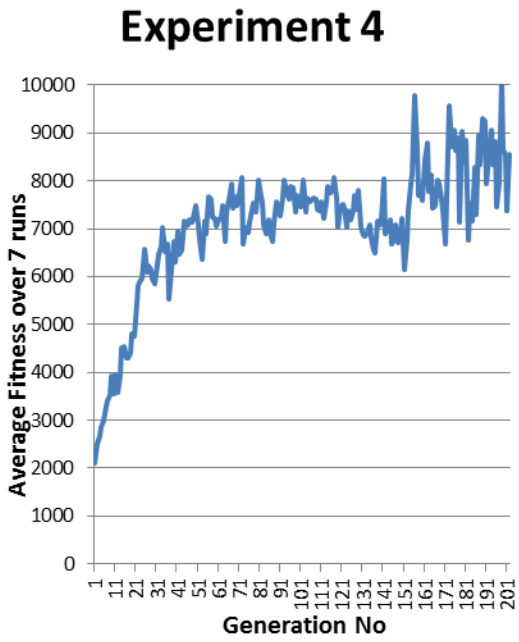

Fig. 5 Average NEAT challenger fitness in experiment 4, 'cycle of counters'

While these variations may seem minor, they demonstrate that the agent could vary its behavior in response to different situations. There were several key differences in the behavior shown by the agent in response to the different champions.

The first difference was the timing and continuance of attack. Against Champion 1, the agent did not issue an early attack order at all, but relied on a defensive stance and the early attack of the opponent to draw its army into the opponent's base. Against Champion 2, the agent attacked early, but ceased the attack and adopted a defensive approach if the attack was unsuccessful. Against Champion 3 NEAT attacked early and continued the attack.

The second difference was in regard to the size and use of the attacking force. Against Champion 1, the agent initially built up a relatively large force, but did not send them into attack. Against Champion 2, the agent immediately attacked with a stream of individual grunts, but then ceased to attack and built up a large army of advanced units before attacking again. Against Champion 3, the agent again immediately attacked with a stream of individual grunts, but in this case continued to press the attack.

As a check that differing strategies were required to defeat the cycle of counters, and to confirm that a single degenerate strategy did not underlie the success of the agent, three sequences of actions, each from a successful game against a different champion were 'frozen' as three separate static AI. These separate static AI were then trialed against each champion from the 'cycle of counters' in a manner similar to experiment 1. In no case was a 'frozen' AI able to defeat all three champions. This demonstrated that adaptive behavior was the key to defeating the cycle of counters.

The behavior demonstrated by NEAT in solving the problem posed by Experiment 4 is associated with complexification of the NEAT ANNs. In the early experiments (1 and 2), a solution was found relatively quickly, and the structure of the neural nets for the solution

were very little different than the starting configuration. However, in experiment 4, the solution took many more generations to evolve, and many of the final networks showed significant differences in structure to the starting configuration. The number of historical markers in successful challengers from experiment 4 was significantly higher than in either experiment 1 or 2.

The agent's modification of strategy in response to the opponent's actions, and in particular the switch from an aggressive to a defensive strategy depending on the success of an attack demonstrated that NEAT can produce agents which adapted to in-game changes without any in-game learning taking place.

IV. Discussion

A. Comparison with Literature:

In [1] integrated agents were created based on an in-depth study of expert player knowledge using a reactive planning language. The AI developed showed similar behaviors to that evolved in the current work in response to the static opponent 'Fast Ogre Rush'. They also found that the 'Knight Rush' strategy (equivalent to our 'Fast Ogre Rush') was easily countered with a Soldier Rush. This strategy is exactly what was observed in the current work against the Fast Ogre Rush in experiment 1.

Ponsen [6] evolved DS scripts showing similar behavior to that shown in this work when countering the 'Soldier Rush' strategy. His DS agent used an optimized Soldier Rush, upgrading grunts to allow the DS agent's army to better the static opponent. In this work agents also used an optimized 'Soldier Rush'. However in this study, the agents used a fast attack rather than upgrading the grunts. Ponsen described a number of counter strategies against the 'Knights Rush' (our 'Fast Ogre Rush'). One identified as an interesting solution was the training of many catapults which was a solution that occurred in the current experiments amongst others.

[11] used CBR to win against 'Soldier Rush' and 'Knights Rush' (equivalent to our 'Fast Ogre Rush'). [12] demonstrated success using a CBR approach against 'Soldier Rush' and 'Knights Rush' (equivalent to our 'Fast Ogre Rush'). The CBR studies are of interest in that they are the only ones that demonstrate effective in-game adaptive behavior. However, the counter strategies have not been reported and thus the results cannot be directly compared. An important aspect of the current work is that it demonstrates that NEAT can also generate adaptive behavior without the extensive database of pre-played games that CBR requires. This might make NEAT more suitable as an adaptive AI technology to support a completely new game.

B. Dominance Tournament Issues

The results of the dominance tournament experiment highlighted an important problem for evolutionary approaches. The problem was that there was no way of measuring how different a successful challenger's strategy was from existing champions, or whether successful challengers were part of the same basin of attraction surrounding a local optimum. This resulted in a proliferation of champions with very similar strategies. In hindsight, it is obvious that sub-optimal solutions might be found, and that micro-incremental improvements in sub-optimal champions would occur. However, the issues involved in resolving these problems are extremely difficult. There are many local optima representing different strategic solutions in the solution space; what is required is a way of identifying these local optima and discarding champions linked to them by gradient ascent. It is only champions representing the peaks of these local optima that should be retained in a champion pool.

Another issue is that it is the nature of a well-balanced RTS game that no one strategy should be able to defeat all other strategies. The notion of a dominance tournament might be counter-productive for evolving agents capable of complex behaviors. Perhaps a stepwise or partial approach might be more appropriate; a first step in which a challenger is only required to beat one or some existing champions, followed by a refinement process where the new solution is optimized before being added to the champion pool.

However, the issues of refining the 'dominance tournament' approach and adapting it to select for complex behavior are subject for future work.

C. Novel Behaviors and Discovery of Exploits:

NEAT agents displayed several novel and unexpected behaviors during early experiments that suggest other applications of the approach beyond creating challenging opponents for human players. These novel behaviors exploited certain limitations of the Wargus environment and highlighted weaknesses in the static AI behaviors. All the problems exposed were addressed before the experiments reported on in this work were performed.

The ability of NEAT to find and exploit weaknesses in the game environment and opposing AI point to its potential utility as a quality assurance tool in RTS AI development.

D. Future Work

1) RTS NEAT Agents Using Partial Knowledge: The input model used for NEAT did not meet the requirements of an ideal AI as identified in [13] because it provided complete knowledge of the opponent's actions and state. Further work could modify this to implement a 'fog of war' and provide agents a partially viewable RTS environment. Such work would investigate whether NEAT can operate as an effective RTS AI under conditions of partial knowledge.

2) Strategic Distance and Gradient Measures: The results of the dominance tournament in Experiment 3 pointed to the need for some way of comparing strategies and whether they belong to the same basin of attraction.

3) Fitness Functions and Guided Search: There is a conceptual issue in rating the fitness of losing strategies. Rating such losing strategies implies expert knowledge on the part of the developer that can guide the evolution of

losing agents towards more successful strategies. However, the point of using GAs is to explore a solution space and potentially discover solutions that the developer did not envisage. Fitness functions can unintentionally restrict and bias the solutions discovered by GA algorithms.

However, it was frequently observed that success in a positively biased configuration acted as a reward stepping stone to developing strategies which could win in the negatively biased configuration. This suggests that using a series of games of gradually increasing difficulty and rewarding only winning strategies could allow successful solutions to be developed without biasing by using fitness functions that attempt to rate the success of losing strategies.

V. Conclusions

Experiment 1 established that NEAT could create agents that effectively countered a single strategy. Experiment 2 showed that if a single simple solution could be found to a complex problem, then NEAT would discover that solution before any adaptive behavior occurred. Similarly in Experiment 3, many small refinements of a simple basic behavior were found, before any significantly different strategies occurred. Experiment 4 recognized this issue and attempted to force evolution of adaptive behavior by preventing the existence of a single solution. Complex behavior did evolve in this experiment, at the cost of significantly greater computational effort involved in discovering a successful agent. Experiment 4 demonstrated that NEAT was capable of generating agents with complex in-game adaptive behaviors.

During development of the framework for the experiments it was also found that adaptive AI were useful in discovering unsuspected flaws in the RTS environment and weaknesses in opposing static AI.

The discovery of multiple different RTS strategies using an evolutionary approach was not as straightforward as initially expected and a simple implementation of the 'dominance tournament' concept is not a viable method for discovering distinctly different RTS strategies. A method of identifying local optima in the solution state space, as well as sub-optimal solutions contained within the 'gradient fields' of those local optima is required.

It has been demonstrated that NEAT can generate agents which show complex in-game adaptive behaviors responding differently in different situations. It proved surprisingly difficult, but placed in a situation where no single solution was possible, NEAT evolved such behavior with no additional AI design input on the part of the developer. This suggests in may be possible to use NEAT to explore possible strategies and responses in new RTS games.

References

[1] 1, McCoy, J., & Mateas, M. (2008). An Integrated Agent for Playing Real-Time Strategy Games. Paper presented at the Proceedings of the 23rd national conference on Artificial intelligence, Chicago, Illinois.

[2] 2, Midtgaard, M., Vinther, L., Christiansen, J. R., Christensen, A. M., & Zeng, Y. (2010). Time-Based Reward Shaping in Real-Time Strategy Games. Paper presented at the Proceedings of the 6th international Adams, E. (2009). Fundamentals of Game Design: New Riders Publishing.

[3] Huo, P., Shiu, S. C.-K., Wang, H., & Niu, B. (2009). Case Indexing Using PSO and ANN in Real Time Strategy Games. Paper presented at the Proceedings of the 3rd International Conference on Pattern Recognition and Machine Intelligence, New Delhi, India. Althoff, K.-D., Bergmann, R., Minor, M., Hanft, A., Sugandh, N., Ontañón, S. (2008). Real-Time Plan Adaptation for Case-Based Planning in Real-Time Strategy Games Advances in Case-Based Reasoning (Vol. 5239, pp. 533-547): Springer Berlin / Heidelberg.

[4] Niu, B., Wang, H., Ng, P. H., & Shiu, S. C. (2009). A Neural-Evolutionary Model for Case-Based Planning in Real Time Strategy Games. Paper presented at the Proceedings of the 22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: Next-Generation Applied Intelligence, Tainan, Taiwan.

[5] Weber, B. G., & Mateas, M. (2009b). A Data Mining Approach to Strategy Prediction. Paper presented at the Proceedings of the 5th international conference on Computational Intelligence and Games, Milano, Italy. http://www.ieee-cig.org/cig-2009/Proceedings/proceedings/papers/cig2009_020e.pdf

[6] Ponsen, M. (2004). Improving Adaptive Game AI with Evolutionary Learning. Master of Science, Delft University of Technology. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.6055&rep=rep1&type=pdf

[7] Stanley, K. O. (2004). Efficient Evolution of Neural Networks through Complexification. PhD Thesis. The University of Texas at Austin. Retrieved from http://nn.cs.utexas.edu/?stanley:phd04

[8] Stanley, K. O. (2005). *Evolving Neural Network Agents in the NERO Video Game*. In Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games, Piscataway, NJ:. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.2913&rep=rep1&type=pdf

[9] J. K. Olesen, G. Yannakakis, and J. Hallam, "Real-time challenge balance in an RTS game using rtNEAT," in Proc. IEEE Symp. Comput. Intell. Games, L. Barone and P. Kingston, Eds., Piscataway, NJ, 2008, pp. 87–94.

[10] Stanley, K. O., & Miikkulainen, R. (2002). The Dominance Tournament Method of Monitoring Progress in Coevolution. Paper presented at the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002) Workshop Program, San Francisco. http://nn.cs.utexas.edu/?stanley:gecco02ws

[11] W.Aha, D., Molineaux, M., & Ponsen, M. (2005). Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game Case-Based Reasoning Research and Development (Vol. 3620, pp. 5-20): Springer Berlin / Heidelberg.

[12] McGinty, L., Wilson, D., Weber, B., & Mateas, M. (2009). Conceptual Neighborhoods for Retrieval in Case-Based Reasoning Case-Based Reasoning Research and Development (Vol. 5650, pp. 343-357): Springer Berlin / Heidelberg.

[13] Spronck, P., Sprinkhuizen-Kuyper, I., & Postma, E. (2002). Evolving Improved Opponent Intelligence. Paper presented at the GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation, SCS Europe Bvba. http://ticc.uvt.nl/~pspronck/pubs/EvolvingImprovedOpponentIntelligence.pdf