Evolving Multimodal Behavior Through Subtask and Switch Neural Networks

Xun Li[1] and Risto Miikkulainen[1]

[1]The University of Texas at Austin
xun.bhsfer@utexas.edu

Abstract

While neuroevolution has been used successfully to discover effective control policies for intelligent agents, it has been difficult to evolve behavior that is multimodal, i.e. consists of distinctly different behaviors in different situations. This article proposes a new method, Modular NeuroEvolution of Augmenting Topologies (ModNEAT), to meet this challenge. ModNEAT decomposes complex tasks into tractable subtasks and utilizes neuroevolution to learn each subtask. Switch networks are evolved with the subtask networks to arbitrate among them and thus combine separate subtask networks into a complete hierarchical policy. Further, the need for new subtask modules is detected automatically by monitoring fitness of the agent population. Experimental results in the machine learning game of OpenNERO showed that ModNEAT outperforms the non-modular rtNEAT in both agent fitness and training efficiency.

Introduction

In complex real-world tasks, intelligent agents need to exhibit multimodal behavior. For instance, in robotic soccer keepaway, keeper agents try to complete as many passes to each other as possible while a taker does its best to steal the ball. When a keeper agent controls the ball, it needs to locate its teammates and pass the ball safely. When a teammate controls the ball, it needs to move to open position and prepare to receive passes (Whiteson, Kohl, Miikkulainen and Stone, 2005). Similarly, in an OpenNERO battle, two teams of agents fight to eliminate the enemy team with their weapons. Agents in each team must locate and navigate to enemy agents when they hide behind obstacles, and attack enemy agents when they fall into the weapon's range (Stanley, Bryant, Karpov and Miikkulainen, 2006).

To produce effective multimodal policies, complex tasks are often decomposed into tractable subtasks. Machine learning techniques can then be utilized to learn subtask behaviors. Subtask behaviors combined with an appropriate arbitration mechanism form a complete multimodal policy. The performance of such a policy depends on the division of the original task, the performance of subtask behaviors, and the mechanism for choosing among them. Therefore, an important research goal is to develop methods that decompose complex tasks automatically into tractable subtasks, learn effective behaviors in each subtask, and combine subtask behaviors to produce agents with effective multimodal behaviors.

A variety of methods exist for learning intelligent behaviors based on policy search; one of the most promising such methods is neuroevolution. Neuroevolution has been shown effective in discovering single control behaviors and has been widely applied to various control problems such as multilegged walking (Valsalam, Hiller, MacCurdy, Lipson and Miikkulainen, 2012; Clune, Beckmann, Ofria and Pennock, 2009), automated driving (Kohl, Stanley, Miikkulainen, Samples, and Sherony, 2006; Togelius, Lucas, and Nardi, 2007) and finless rocket control (Gomez and Miikkulainen, 2003). However, it is a challenging task to learn multimodal behaviors through neuroevolution.

One approach is to implement module mutation as part of the algorithm. This approach includes a structural mutation operator that adds a new module to a neural-network-based policy (Schrum and Miikkulainen, 2009). Several methods to implement module mutation have been proposed including Module Mutation Previous (MMP), Module Mutation Random (MMR), and Module Mutation Duplicate (MMD) (Schrum, 2014). Another approach is to use indirect encoding such as HyperNEAT to create the modules. Experimental results indicated that HyperNEAT is able to produce modular solutions rapidly for simple tasks that require multimodal behaviors (Clune, Beckmann, McKinley and Ofria, 2010). Various indirect encoding approaches can also be used to incorporate multimodality better in HyperNEAT (Pugh and Stanley, 2013). On the other hand, hierarchical learning approaches such as concurrent layered learning and coevolution were shown to work well in robotic keepaway soccer using neuroevolution with Enforced Sub-Populations (ESP). However, in the full robocup soccer simulator, homogeneous agents controlled by monolithic networks performed the best (Subramoney, 2012).

The prior studies concentrated on structural modification of neural-network-based policies, i.e. the initial structure and weights of new modules and how these modules could be best combined with existing modules. Thus, they did not address the problem of how such drastic modifications should be timed. Moreover, they did not demonstrate how proper arbitration between the subtasks could be discovered while evolving the subtask networks. This article proposes Modular NeuroEvolution of Augmenting Topologies (ModNEAT) as a solution to these problems. ModNEAT detects the need for module mutation by monitoring population fitness. Experimental results in the machine learning game of OpenNERO showed that ModNEAT is effective in finding appropriate task division and learning multimodal behaviors for complex tasks.

The remaining sections of this article are organized as follows: the next section introduces the ModNEAT algorithm, the third section presents and analyzes experimental results based on OpenNERO, and the fourth section reviews key ideas of this article and points out directions for future work. Following that is a brief conclusion.

Method

Modular NeuroEvolution with Augmenting Topologies (ModNEAT) is an evolutionary computation algorithm for learning multimodal policies with minimum human guidance. In this section, task decomposition and policy structure in ModNEAT are first introduced, followed by a description of the NEAT algorithm used for evolving the switch network and the subtask networks in a multimodal policy. The mechanism for adjusting the probability of module mutation based on population fitness is introduced next, and evolution of subtask and switch networks is discussed in the end.

Task Decomposition

To learn multimodal behaviors effectively, complex tasks are often decomposed into multiple subtasks (Jain, Subramoney and Miikkulainen, 2012). Machine learning techniques are applied to each subtask, and subtask policies combined to form a complete multimodal policy. In a neural-network-based multimodal policy, subtask policies are represented by different neural networks. Various mechanisms including human-specified rules, decision trees, and switch networks can be used to arbitrate among subtask policies (Whiteson, Kohl, Miikkulainen and Stone, 2005).

This approach is beneficial for two reasons. First, neural networks that are competent in one subtask may need to be adapted significantly to perform well in another subtask. By training neural networks separately for each subtask, selection in evolution becomes more focused, improving training efficiency and resulting in better agent behaviors. Second, task decomposition allows agents to learn complicated behaviors by combining behaviors from different subtasks. Consequently, agents can achieve high fitness in a variety of complex situations.

Whereas subtasks are usually defined based on human knowledge, ModNEAT performs task decomposition automatically. The method to detect the need for initiating new subtasks is introduced in the third subsection.

To allow greater flexibility for evolution, ModNEAT uses switch networks to combine subtask solutions (Figure 1). The complete policy of an agent contains: (1) *n* subtask networks that receive identical inputs, and (2) a switch network that receives the same inputs as the subtask networks and arbitrates among subtask networks, i.e. to decide which subtask network takes control of the agent during the current time step.

Since both switch networks and subtask solutions are neural networks, all components of a multimodal policy can be evolved through neuroevolution. In ModNEAT, fitness is defined based on the performance of the complete agent. The genetic encoding of the switch network and one or more subtask networks are concatenated into a single genome and evolved using the NEAT neuroevolution method as will be described next.

NeuroEvolution of Augmenting Topologies (NEAT)

Neuroevolution of Augmenting Topologies (NEAT) is a genetic algorithm for evolving artificial neural networks (Stanley and Miikkulainen, 2002). The algorithm for starts with an initial population of random networks. A fitness function based on domain knowledge is used to evaluate the neural networks in all generations. Those with the highest fitness are selected as parents of the next generation. Through mutation and crossover among the parents, a new generation of networks is created, and the next iteration of evolution begins. This process repeats until a satisfactory solution is found.

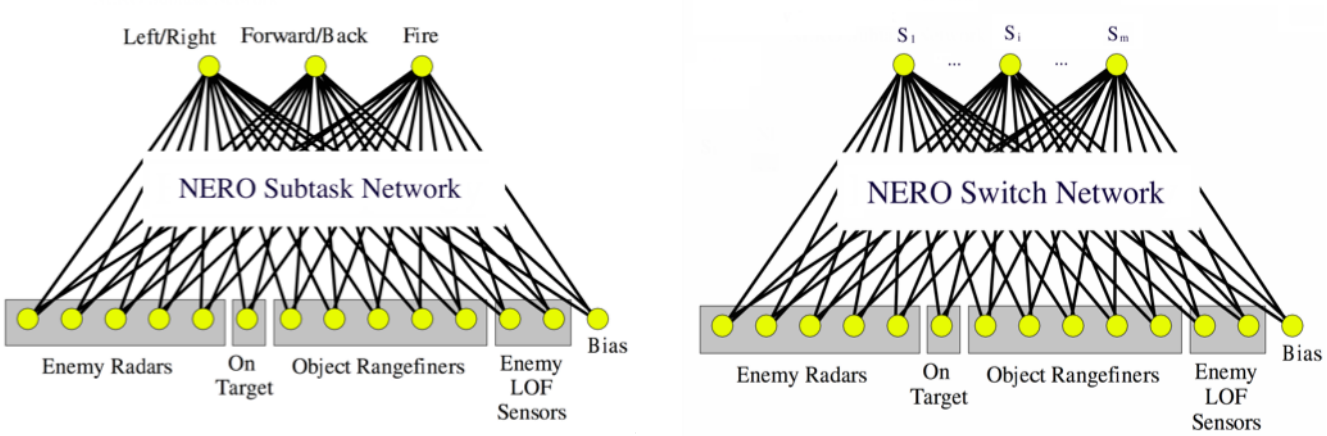


Figure 1: Subtask and Switch Network in the NERO Domain – The network on the left is one of the *m* subtask networks in the complete solution, and the network on the right is the switch network. Nodes in the bottom represent inputs from the environment, and S_1 to S_m are the outputs of the switch network. Each output of the switch network specifies the preference, for each of the *m* subtask networks, given current inputs. The output of the subtask network with the maximum weight is chosen as the output of the whole system. In this manner, the switch network selects the appropriate subtask network for each input situation.

In NEAT, mutation and crossover are not limited to modifying the weights of the neural networks. Topological augmentation is allowed so that various topologies can evolve, providing more flexibility for the genetic algorithm. Neural networks in NEAT are adapted in four ways:

1. Weights of one or multiple links are modified.
2. Weights are crossed over.
3. A link is added between two existing nodes.
4. A node is inserted in the middle of an existing link.

Note that modifications 3 and 4 are both topological augmentations. They allow the algorithm to start with relatively simple initial networks and gradually increase their topological complexity.

To protect innovation, when a new link is added between two existing nodes, the initial weight of the new link is close to zero. Similarly, when a new node is added, the in-link to the new node has a weight that is close to one, and the out-link from the new node preserves the weight value of the original link. Consequently, the initial impact of topological augmentation is limited, and networks with new structures are more likely survive in the evolution process.

In addition, to guarantee reasonable crossover between genotypes, an innovation number (global ID) is assigned for each new structure. For instance, when a new node is inserted to an existing link, the ID of the original link becomes obsolete, and a new ID is assigned to the each of the in-link, out-link, and the new node. Each genotype maintains a list of IDs for its structures (nodes and links), and crossover is limited to structures with identical IDs.

In order to protect innovation, NEAT speciates the population based on genotype similarity, which is a weighted sum of similarity score of weighting parameter values and network topological structures. Network fitness is compared primarily within species so that networks with innovations are offered enough time for optimization. Explicit fitness sharing (Goldberg and Richardson, 1987) is used for fitness evaluation so that species size is limited and no single species can easily take over the entire population.

In game domains, generation changes could result in noticeable discontinuations. Therefore, in the rtNEAT version of NEAT (Stanley, Bryant, and Miikkulainen, 2005), agents evolve while maintaining constant interaction with task environment. Instead of updating generations of populations, rtNEAT generates offspring continuously, one at a time. Thus, evolution is gradual and less visible to the human observers. Since ModNEAT experiments were run in the OpenNERO games, they were based on the rtNEAT method as well.

Automatic Subtask Initiation

One key issue in task decomposition is to detect the need for subtask initiation, i.e. to figure out when it is necessary to insert the appropriate structure (i.e. subtask networks and switch networks) to the current policy so that a new subtask can be initiated.

In ModNEAT, the need for subtask initiation is detected automatically by monitoring population fitness in consecutive generations. Intuitively, when the standard deviation of agent fitness remains low in multiple consecutive generations (i.e. evolution is not able to produce significant variation any longer), additional subtask networks may be needed. The probability of subtask initiation in ModNEAT is defined as:

$$p_s = \begin{cases} 0, & n_i < N \\ \frac{(n_i - N)}{N}, & N \le n_i < 2N \\ 1, & n_i \ge 2N, \end{cases} \qquad (1)$$

where p_s is the probability of initiating subtask networks, n_i is the number of consecutive *ineffective generations* after the ith iteration of evolution, and N is an impatience parameter. When the evolution process is effective, p_s remains zero; when evolution falls into a performance plateau, the probability of subtask initiation increases linearly to one. Ineffective generation is defined as one where the standard deviation of agent fitness is less than a threshold parameter μ. Note that while multiple generations may be ineffective during evolution process, only consecutive ineffective generations may result in higher probability of subtask initiation.

Solution Evolution

ModNEAT starts with a population of simple initial policies. These policies can be either randomly generated or human-specified. Each initial policy contains a single subtask network and an initial switch network. The switch network contains only one output and grows as more subtask networks are added to the policy.

When a new subtask is initiated, a randomly initialized subtask network is added to the policy, and a new output node is added to the switch network. Three measures are taken to protect innovations:

1. Solution Complexity Control
   When a new subtask network is added to a genotype, no other subtask networks can be added to the same genotype in the next N (the impatience parameter in equation 1) generations of evolution. Solution Complexity Control prevents unnecessary insertion of subtask networks and limits the complexity of the complete solution.
2. Prior Feature Protection
   All existing subtask networks and switch network structures of a genotype are frozen in the next N iterations of evolution if a new subtask network is added to the genotype. In this manner, existing features that are advantageous in other subtasks are protected.
3. Population Speciation
   As is mentioned in the previous subsection, population speciation is used to protect innovations. Since similarity scores between genotypes with different number of action subtask networks tend to be low, genotypes that contain additional subtask networks are usually regarded as a new species and are protected by speciation, which provides more time for new subtask networks to evolve.

Throughout the evolution process, fitness is evaluated for the policy as a whole. All modules, i.e. subtask networks and the switch network, are encoded in a single concatenated genome and therefore evolved together. Such evolution encourages subtask networks and switch networks that work well together to emerge. In addition, evolution combined with automatic subtask initiation encourages more complicated behaviors by combining existing subtask

behaviors to emerge. It is less likely for new subtask networks to be needed as the number of existing subtask networks increases. Thus, the overall complexity, i.e. the number of subtask networks in the multimodal policy, remains limited.

Experiments

This section presents experimental results based on the open source machine learning game OpenNERO to evaluate the effectiveness of ModNEAT. The first subsection introduces the OpenNERO platform, the second describes experimental setup, and the third compares ModNEAT with rtNEAT in both training efficiency and agent performance.

OpenNERO Platform

To test the effectiveness of ModNEAT, the performance of game agents generated by rtNEAT and Mod NEAT were compared in the OpenNERO implementation of the NERO machine learning game (opennero.googlecode.com; Karpov, Sheblak, and Miikkulainen, 2008; Stanley, Bryant, and Miikkulainen, 2005). In NERO, players design training schemes for a team of game agents. To defeat the enemy team, game agents must be able to target enemy agents and shoot them with their weapons. Since the arena contains obstacles such as walls and trees, agents need to be able to locate the enemy agents and navigate to them.

In the training mode of NERO, players set up training sessions by adding enemy turrets, flags, and walls. Also, game agents can be spawned at different locations, and all objects (flags, turrets, walls) can be moved around the arena. To evaluate fitness of the agents, players can set reward value for various behaviors such as sticking together, hitting enemy turrets, and getting closer to flags or enemy turrets. The overall fitness of an agent is a weighted sum of the rewards it obtains in each fitness measurement. A team of agents can be trained through a sequence of training sessions with different environment layouts and reward settings.

In the battle mode, two teams of agents are spawned on two sides of the middle wall in the arena. Each team consists of 50 game agents. Every agent has 20 hit points at the beginning. Whenever hit by enemy weapon, an agent loses one hit point. An agent is removed from the battlefield once it loses all hit points. The team that eliminates all enemy agents wins.

OpenNERO is an open-source version of NERO developed as a research and education platform. IT duplicates the original NERO game with a few minor differences on how the sensors and actions are implemented. In OpenNERO, every agent has 18 inputs excluding the bias:

1. Laser range finders that sense distance to the nearest obstacle (wall or a tree) in five directions: front, 45 degrees to each side, and 90 degrees to each side.
2. Radar sensors that return the distance to the flag in 5 overlapping sectors: -3 to 18, 12 to 90, -18 to 3, -12 to -90, and -90 to 90 degrees (behind the agent).
3. Radar sensors that trigger when enemies are detected within 5 sectors of the space (as defined above) around the robot. The more enemies are detected or the closer they are, the higher the value of the corresponding radar sensor will be.
4. Two sensors that depend on the distance and direction to the center of mass of the teammates.
5. A sensor that indicates whether the agent is facing an enemy within sensor range within 2 degrees.

The outputs of the control policy of an agent are:

1. Forward/backward speed: -1 to 1 of maximum. The agents can move at a rate of up to one distance unit per frame.
2. Turning speed: -1 to 1 of maximum. The agents can turn at a rate of up to 0.2 radians per frame.

Note that there is no output for taking shots. Instead, the agents shoot probabilistically. To attack an enemy, an agent must be oriented within 2 degrees of the target. Shooting accuracy increases linearly so that at 2 degrees an agent has a 50% chance of hitting, and if it face the center of the target exactly, it always hits. Also, the agent must be closer than 600 distance units from the target. Between 600 and 300, shooting likelihood increases linearly, and within 300, it always shoots.

OpenNERO provides a suitable platform to evaluate the performance of ModNEAT – the complex task of defeating the enemy team can be accomplished by combining behaviors in different subtasks such as navigating to enemy agents, aiming and shooting, and these behaviors can be learned sequentially through training sessions.

Experimental Setup

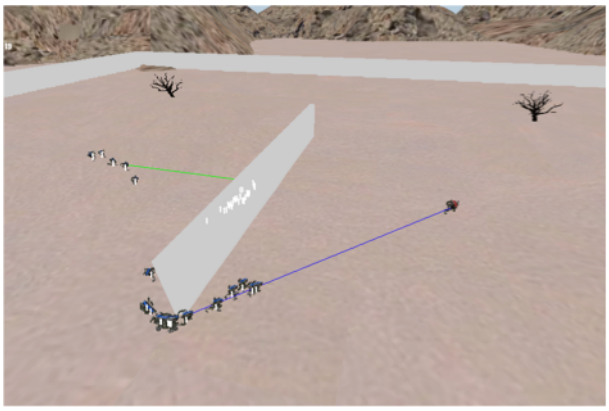
To compare the performance of ModNEAT with rtNEAT, two teams were trained and evaluated based on training time and tournament score. In order to ensure fair comparison, both teams were trained through identical training sessions. The objectives of each training session are listed in table 1. The training proceeded from S1 to S5, and a training session was terminated if average fitness of network population reached a preset value, or training time exceeded an hour.

In S1, a single enemy turret was placed in the arena, and there was no obstacle between spawning point and the turret. In S2, the enemy turret slowly moved around the spawning point to simulate moving enemy agents. In S3, an enemy turret was placed behind the wall (but far away from it), and agents were expected to go around the wall and hit it. In S4, enemy turrets were not only behind the wall but also moving around. In S5, enemy turrets were placed behind and close to the wall. Note that S5 is particularly difficult because agents must be able to tell if the enemy is in front of or behind the wall. If the enemy is behind the wall, agents should first go around the wall and then stop to shoot at the enemy.

Session	Objective
S1	Hitting still enemy in sight
S2	Hitting moving enemy in sight
S3	Hitting still enemy behind the wall
S4	Hitting moving enemy behind the wall
S5	Hitting enemy that hides closely behind the wall

Table 1: Training Sessions

(a) rtNEAT



(b) ModNEAT

Figure 2: Training Snapshots in S3 – Most rtNEAT agents (a) kept shooting at the wall. The green lines indicate that an obstacle blocks the shot. In contrast, most ModNEAT agents (b) managed to go around the wall using the new subtask network, i.e. subtask network 1. The white number on a ModNEAT agent is the ID of the subtask network that is currently active. They are using a different subtask network to get around the wall and shoot at the enemy.

In all of the above sessions, fitness reward was set to 100 for hitting enemy turrets. In S1, S2 and S5, rewards were 0 for all other possible behaviors (i.e. sticking together, approaching the flag or enemy turrets, etc), while in S3 and S4 reward for getting closer to enemy turret was set to 10 (all others were 0).

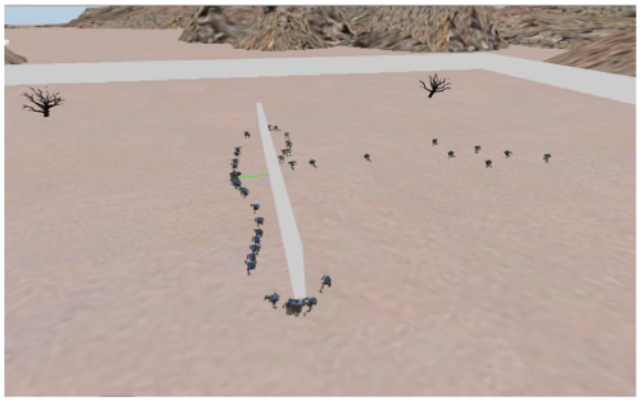
Results and Analysis

Table 2 shows the training time for each of the two teams. S3 is marked red because ModNEAT initiated most subtasks in this session, which significantly reduced the training time. The reason is that S3 requires significantly different network structures from S1 and S2, and it is difficult for rtNEAT to adapt networks that are trained for aiming and shooting at in-sight enemies to accomplish the task of reaching enemies behind the wall. As a matter of fact, in the first 25 minutes of S3 using rtNEAT, most of the agents simply kept shooting at the wall or running into it. In comparison, game agents trained with ModNEAT managed to bypass the wall soon with the help of a new subtask network (Figure 2).

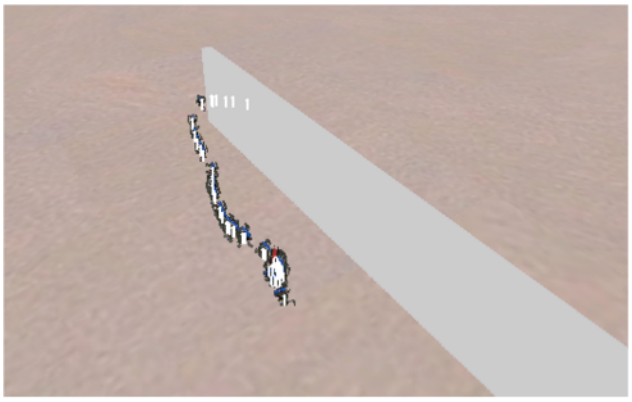
Session	rtNEAT	ModNEAT
S1	5	5
S2	5.5	6
S3	48.5	28.5
S4	7	8
S5	60+	12
Total	126+	59.5

Table 2: Training Time (minute)

Another important observation is that in session 5, the rtNEAT team never managed to reach satisfactory average fitness score while the ModNEAT team reached fitness goal within 12 minutes, even without introducing additional subtask networks. The reason can be found in the structure and function of the subtask networks.



(a) rtNEAT



(b) ModNEAT

Figure 3: Training Snapshots in S5 – Most rtNEAT agents (a) kept moving around the wall over and over again, i.e., they did not stay on the enemy's side to engage it. In contrast, ModNEAT agents (b) first navigated to the enemy turret using subtask network 1. When getting close, ModNEAT agents started attacking the enemy turret using subtask network 0. ModNEAT agents also learned to stay close to the enemy because they were more likely to hit the enemy turret when the distance is shorter.

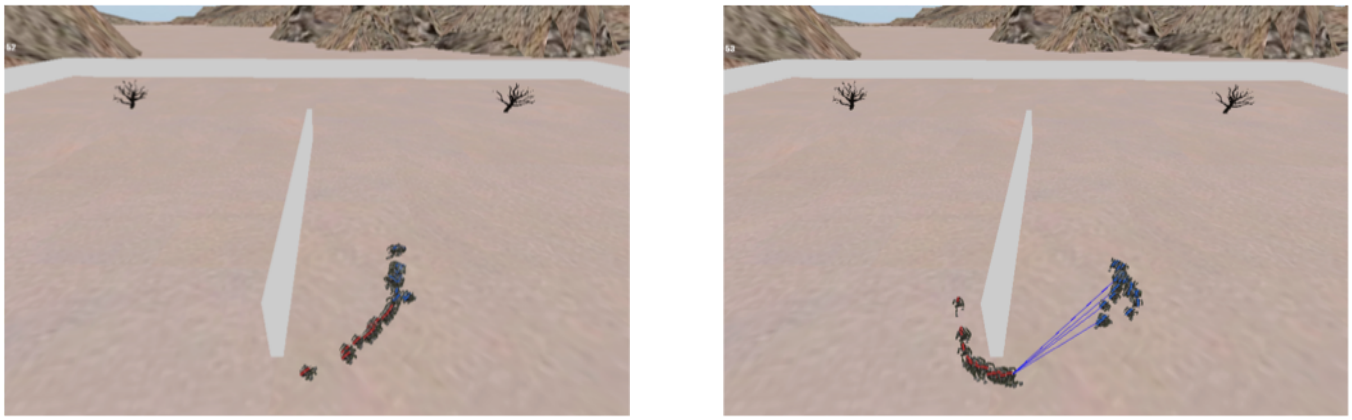


Figure 4: Snapshots of a Battle between rtNEAT (red) and ModNEAT (blue) – This series of two snapshots shows how the ModNEAT agents exploit weakness of the rtNEAT agents. In the left picture, the two teams ran into each other from opposite directions. The rtNEAT agents continued marching around the wall after the encounter, but the ModNEAT agents start turning around to attack the enemies. The right snapshot is taken a few seconds later. After turning around, the ModNEAT team deals considerable damage to the rtNEAT team.

Team	ModNEAT	rtNEAT	me-rambo	synth.pop	synth-flag.pop	coward1
Max Score	649	542	512	409	398	359
Min Score	216	-649	-359	-542	-512	-471
Avg Score	424.1	253.5	187.5	82.09	68.45	13.18
Wins/Losses	11/0	10/1	8/3	8/3	7/4	6/5

Table 3: Agent Performance – Table 3 shows the performance of the ModNEAT team, the rtNEAT team, and the top four teams from the 2011 OpenNERO Tournament. The score of a team in a battle was the remaining hit points in the entire team minus that of the opponent team. Team A defeats team B in a match if the sum of A's two battle scores with B is positive. Wins/Losses indicates the number of matches a team won/lost.

For all ModNEAT agents whose final fitness was above the threshold, the complete solution consisted of two subtask networks. In session S1 to S4, subtask network 1 was used to get close to the enemy if it was far away. Subtask network 0 was then used to aim at the enemy and approach the enemy slowly while shooting. In session S5, when enemy turrets were placed right behind the wall, ModNEAT agents activated subtask network 1 to get around the wall. After getting close to the enemy, they switched to subtask network 0 to shoot the enemy. Thus, complex behaviors of reaching and eliminating well-hidden enemies were acquired by combining simple behaviors in different action modes.

In contrast, rtNEAT agents have only a single solution network which attempts to integrate both of the above functionalities. To reach enemy turrets right behind the wall, solution networks must learn to move around the wall. To deal enough damage to enemy turrets, solution networks must learn to aim at the enemy and focus on shooting. Consequently, rtNEAT evolved a compromise policy where agents keep moving around the wall and shoot enemy turrets on the way. Even though they eventually evolved to reach the enemy turrets, rtNEAT agents wasted too much time moving around the wall and therefore never managed to reach the fitness threshold for S5 (Figure 3).

In the complete solution of the ModNEAT agents, each subtask network contained 18 inputs and two outputs (as specified in the OpenNERO Platform subsection). Subtask network 0 of the champion agent in the last generation contained six hidden nodes and 64 links, subtask network 1 contained eight hidden nodes and 72 links, and the switch network had four hidden nodes and 52 links. Note that most hidden nodes are not fully connected to the inputs and outputs, and all networks were non-recurrent. On the other hand, the solution network of the rtNEAT champion is a single network with 10 hidden nodes, 78 links, and the same number of inputs and outputs. Such network structures are typical for agents evolved by rtNEAT.

In order to evaluate the performance of the two teams, both teams were tested in a tournament against the top 10 teams in the 2011 OpenNERO Tournament (code.google.com/p/opennero/wiki/TournamentResults2011). Every team participated in 11 matches to compete with all 11 opponents. Each tournament match is consisted of two battles (with the spawning location switched). The results of the tournament are shown in Table 3.

While both of the two teams defeated all ten tournament teams, the ModNEAT team defeated the rtNEAT team by a large margin, i.e. by 649 hit points. The agents trained with rtNEAT (as well as the agents in many top-ranked teams in the 2011 OpenNERO Tournament) tended to keep moving around the wall over and over again after encountering the enemy. The ModNEAT agents exploited this weakness by switching to shooting mode while the enemy agents were busy moving ahead (Figure 4). Thus, the subtask and switch network architecture allowed evolving distinctively better behaviors than before.

Discussion and Future Work

Modular NeuroEvolution of Augmenting Topologies (ModNEAT) is an evolutionary computation method aimed at tackling complex tasks with minimum human guidance. The algorithm detects the need for adding additional subtask networks automatically by monitoring variation in population fitness. Switch networks are employed to arbitrate between subtask networks based on inputs from the environment. The subtask networks and switch networks are evolved through NEAT-based evolution together in a single genome. Experimental results showed that compared to rtNEAT, ModNEAT generates agents with significantly higher performance in shorter training time.

ModNEAT has higher training efficiency because it does not attempt to adapt networks trained for certain objectives to accomplish new objectives if they require substantially different network features (i.e. topological structure and link weights). Instead, new subtask networks are initiated and evolved to reach such objectives.

ModNEAT manages to learn more intelligent behaviors because task decomposition allows natural selection to be more focused on each subtask, and thereby makes subtask behaviors more robust and effective. In addition, ModNEAT has higher potential to learn intelligent behavior that is difficult to learn with a single network because switch networks allow relatively simple subtask behaviors to be combined flexibly to exhibit more complex behaviors.

An interesting direction of future work is to combine ModNEAT with various module-mutation techniques (Schrum, 2014) to develop more effective methods for initiating, evolving, and combining subtask networks. Another interesting idea is to explore other fitness-based mechanisms to adjust subtask initiation probability so that better task division can be found with fewer attempts. Third, ModNEAT can be applied to many challenging problem domains such as robot soccer and Non-player Character (NPC) design for video games to produce intelligent agents with effective multimodal behavior.

Conclusion

Complex real-world tasks often require intelligent agents to exhibit multimodal behavior. Although neuroevolution has been successfully used in learning single control behavior, it has been difficult to develop multimodal behavior in this approach. This article shows how such behavior can be constructed by discovering and evolving subtask networks with a switch network, i.e. using the ModNEAT approach. The results in the interactive game of OpenNERO show that ModNEAT is effective, i.e. it results in better policies and evolves faster than standard neuroevolution. Thus, ModNEAT can be useful in constructing intelligent agent behaviors for games and robotics in the future.

Acknowledgements

This research was supported in part by NSF grants DBI-0939454 and IIS-0915038, and in part by NIH grant R01-GM105042.

References

Whiteson, S., Kohl, N., Miikkulainen, R., Stone, P. (2005). Evolving Keepaway Soccer Players through Task Decomposition. *Machine Learning*, 59(1):5–30.

Stanley, K. O., Bryant, B. D., Karpov I., Miikkulainen, R. (2006). Real-Time Evolution of Neural Networks in the NERO Video Game. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*,1671–1674, Boston, MA.

Valsalam. V. K., Hiller, J., MacCurdy, R., Lipson, H., Miikkulainen, R. (2012). Constructing Controllers for Physical Multilegged Robots using the ENSO Neuroevolution Approach. *Evolutionary Intelligence*, 5(1):1–12.

Clune, J., Beckmann, B.E., Ofria, C., Pennock, R. T. (2009). Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC)*, Trondheim, Norway.

Kolh, N., Stanley, K. O., Karpov I., Miikkulainen, R., Samples, M., Sherony, R. (2006). Evolving Real-world Vehicle Warning System. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 1681-1688, ACM, New York, NY, USA.

Togelius, J., Lucas, S., and Nardi, R. (2007). Computational Intelligence in racing games. In: *Advanced Intelligent Paradigms in Computer Games*, pp. 39-69. Springer.

Gomez, F. J., Miikkulainen, R. (2003). Active Guidance for a Finless Rocket Using Neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2084-2095, San Francisco. MorganKaufmann.

Schrum, J., Miikkulainen, R. (2009). Evolving Multimodal Behavior in NPCs. In *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, 325-332, Milan, Italy.

Clune, J., Beckmann, B. E., McKinley, P. K., Ofria, C. (2010). Investigating Whether HyperNEAT Produces Modular Neural Networks. In *Proceedings of the Genetic and Evolutionary Computation* Conference, 635-642.

Pugh, J. K., Stanley, K. O. (2013). Evolving Multimodal Controllers with HyperNEAT. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*, 735-742, New York, NY: ACM.

Subramoney, A. (2012). Evaluating Modular Neuroevolution in Robotic Keepaway Soccer. Master Thesis, Department of Computer Science, the University of Texas at Austin, Austin, TX.

Jain, A., Subramoney, A., Miikkulainen, R. (2012). Task Decomposition with Neuroevolution in Extended Predator-Prey Domain. In *Proceedings of Thirteenth International Conference on the Synthesis and Simulation of Living Systems*, East Lansing, MI.

Stanley K. O., Miikkulainen, R (2002). Evolving Neural Networks Through Augmenting Topologies. *Evolution Computation*, 10(2): 99-127.

Goldberg, D. E., Richardson, J. (1987). Genetic Algorithms with Sharing for Multimodal Function Optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, page 148-154, Morgan Kaufmann, San Francisco, CA.

Stanley, K. O., Bryant, B. D., Miikkulainen, R. (2005). Evolving Neural Network Agents in the NERO Video Game. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*. Piscataway, NJ: IEEE.

Karpov, I. V., Sheblak, J., Miikkulainen, R. (2008). OpenNERO: a Game Platform for AI Research and Education. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*.

Schrum, J. (2014). Evolving Multimodal Behavior Through Modular Multiobjective Neuroevolution. PhD Thesis, Department of Computer Science, the University of Texas at Austin, Austin, TX.