UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER

COMPUTER SCIENCE HONOURS
FINAL PAPER
2015

Title: An empirical evaluation of evolutionary controller design methods for collective gathering task

Author: Jae Jang

Project Abbreviation: ASCIRT

Supervisor: Dr. Geoff Nitschke

Category	Min	Max	Chosen
Requirement Analysis and Design	0	20	0
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	15
System Development and Implementation	0	15	10
Results, Findings and Conclusion	10	20	20
Aim Formulation and Background Work	10	15	15
Quality of Paper Writing and Presentation	10		10
Adherence to Project Proposal and Quality of Deliverables	10		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	0	10	
Total marks		**80**	**80**

An empirical evaluation of evolutionary controller design methods for collective gathering task

Jae Jang
Department of Computer Science
University of Cape Town
Cape Town, South Africa
Email: jngjae001@myuct.ac.za

ABSTRACT
This research aims to evaluate the performance of evolutionary controller design methods for developing a collective behaviour for a team of robots. The methods tested in this research are NEAT which is capable of finding minimal solution quickly, and SANE which maintains high genetic diversity through neuron level evolution. The task chosen for these methods was a collective gathering task which required a team of robots to cooperate in finding and retrieving item of interest. Our results showed that NEAT consistently produced better controllers compared to SANE.

Keywords
Evolutionary robotics; Multi-agent system; NEAT; SANE

1. INTRODUCTION
Over the past decades robotics research focused on creating *autonomous mobile robots* [28]. These robots are composed of three major components. The first component is the sensory system responsible for perceiving the surrounding environment. The next component is the motor system which allows the robot to move around in the environment and execute actions. The final component is the *controller* which maps the inputs received by the sensory system to the appropriate actions, thus defining the behaviour of the robot. Well-designed autonomous mobile robots can perform various actions to achieve given objectives without any direct human intervention.

Multiple robots can be deployed in the same environment to create an even more powerful robotic system. *Multi-agent system* (MAS) is a field concerning coordination of behaviours of multiple robots so that global joint behaviour can be achieved [5]. Using multiple agents has the benefit of providing redundancy and larger geographical coverage, but more importantly MAS allows team of robots to cooperate with each other to develop a *collective behaviour*. This emergent behaviour can be used to solve challenging tasks that cannot be solved by a single robotic agent. An effective multi-agent system can be useful at solving real world problems in remote or hazardous environments which are not suitable for human workers.

Creating an effective robotic system often requires close integration with another field of study called artificial intelligence [23]. *Artificial intelligence* (AI) is a discipline which aims to create computer systems capable of making intelligent decisions without human control. Entities equipped with artificial intelligence are referred to as *agents*. *Machine learning* (ML) is a sub field of AI which focuses on improving the performance of agents with experience (process also known as *learning*) [16]. Traditionally agents were hand designed using extensive knowledge regarding the agent, task and the environment. Machine learning reduces the complexity of the design process by allowing imperfect agents to adapt its behaviour to satisfactory standard, rather than requiring a complete functional agent from the start. This paper investigates the performance of such learning methods on designing a controller for a multi-agent system.

1.1 Machine Learning
Machine learning can be divided into three main categories based on the type of feedback provided to the computer system.

1.1.1 Supervised Learning
In *supervised learning*, agents are provided with input-output pair of examples from external supervisor [14]. Through training, agent learns to reproduce the explicit behaviour defined by such training data. Learning from Demonstration (LfD) [2] is a supervised technique that can teach robots to learn a policy that maps current environmental state to appropriate actions by providing examples. LfD has been successfully applied to robotic applications such as robot soccer, maze navigation, etc. [1]. However as with all supervised learning approaches, it has the limitation of requiring large volume of high quality data to create an effective learning mechanism.

1.1.2 Unsupervised Learning
Learning problems where there is no explicit target output provided for a given input is called *unsupervised learning* [4]. In this case the goal of the agent is to learn some form of pattern from given inputs. Unlike supervised learning this approach does not require any training data, which makes it a suitable for training complex robotic systems.

1.1.3 Reinforcement Learning
In *Reinforcement learning* (RL) [3] every action executed at a specific state is followed by a numerical scalar feedback that can be interpreted as either a reward or a penalty. Unlike supervised learning, agent develops a policy through a continual trial and error interaction with the environment, where agent uses a policy combined with randomness to determine the action to be executed for a given state. For each state, actions that resulted in positive feedback are encouraged while actions that resulted in negative feedback are discouraged. Overtime agent's policy is refined using past experience so that maximum aggregate reward can be achieved.

1.2 Evolutionary Computation
Evolutionary computation (EC) [7] is a research area that uses the principles of natural evolution to solve search or optimization problems. *Evolutionary Algorithm* (EA) [8] is a subset of EC which uses the theory of natural selection as an underlying idea to refine a population of potential solutions. Each member of the

population has a genome (genetic chromosome) that defines their characteristics as a solution. The population is evolved iteratively by refining the genomes of the population through bio-inspired genetic mechanisms such as *natural selection*, *recombination* and *mutation* until a specific stopping condition occurs Selection operator is responsible for selecting high fitness (evaluation of a performance of a solution) individuals for reproduction, while recombination (crossover) operator is responsible for producing offspring by combining the genetic chromosome of its parents. The mutation operator randomly modifies a genome of an individual to potentially discover unexplored solution. Different variations of EAs have been developed defined by their genetic representations and search operators. Popular examples of EAs include *Genetic Algorithm* (GA) [13], *Genetic Programming* (GP) [15], *Evolutionary Programming* (EP) [10] and *Evolution Strategies* (ES) [21].

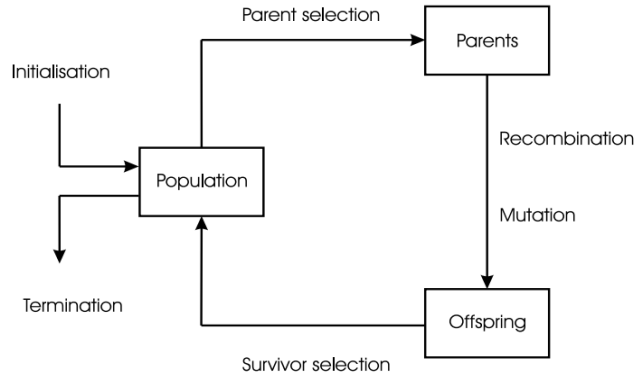

Figure 1. The general scheme of an Evolutionary Algorithm as a flow-chart.

1.3 Controller Design

Controller design can be seen as a sub discipline of AI which focuses on making intelligent agents by creating an artificial controller for simulated (software) and embodied (robotic) agents. Traditional approaches to controller design followed a divide and conquer principle by dividing the desired global behaviour into hopefully simpler sub-behaviours. An agent's control system was also divided into sub-components, with each component in charge of a single sub-behaviour. The Global behaviour of the agent was achieved through coordination between the sub-components [19].

This approach of controller design becomes extremely challenging when designing a controller for a dynamic environment. In dynamic environment the global behaviour of an agent will be the emergent result of dynamic interaction between the controller's sub-components and the environment. Since accurately predicting the result of dynamic interaction is extremely difficult, designer is not able to figure out which sub-components are required to produce the target global behaviour. This is also known as the *design problem*. To address this issue research on controller design has focused towards using machine learning approaches.

1.4 Artificial Neural Network

An *Artificial Neural Network* (ANN) is a computational model which was created in an attempt to mimic the way in which biological brains processes information [9]. It is often used as a controller for a robotic agent as they possess features such as robustness to noise and fault tolerance. The basic structure of ANN consists of layers of interconnected artificial nodes. Nodes and their interconnections are analogous to neurons and synapses of a human brain. The connections between nodes are weighted by a numerical value which modifies the value going through it. Each node can be seen as a processing unit that computes its output by performing a transfer function on the weighted sum of its inputs [29]. Weighted summation simply sums up the product of the input and the connection value from the input's source. Following equation is an example of a summation function for node *i* which receives inputs from node *j*. Weight values are labelled *w* and input value are labelled *x*.

$$\sum_{j=1}^{n} w_{ij} x_j \qquad \textbf{(1)}$$

The value obtained from the summation function is used as input for the transfer function which determines the output of the neuron. Transfer function is typically a non-linear, differentiable function such as the sigmoid function shown below.

$$f(z) = \frac{1}{1+e^{-z}} \qquad \textbf{(2)}$$

Computation in ANN is done layer by layer where output from each node is sent to the connected nodes in the next layer. These values in turn act as input to the transfer function of the receiving node. This process repeats until the output from the final layer is emittted.

When used as a controller for a robot agent, the first layer acts as the input layer which receives readings from the sensory system. These values are sent to the nodes in the next layer, also known as the hidden layer. Hidden layer is responsible for performing computation on the received input value and it once again sends its output to the next and the last layer (output layer). The computed value from the output layer determines the final action to be performed for the given input.

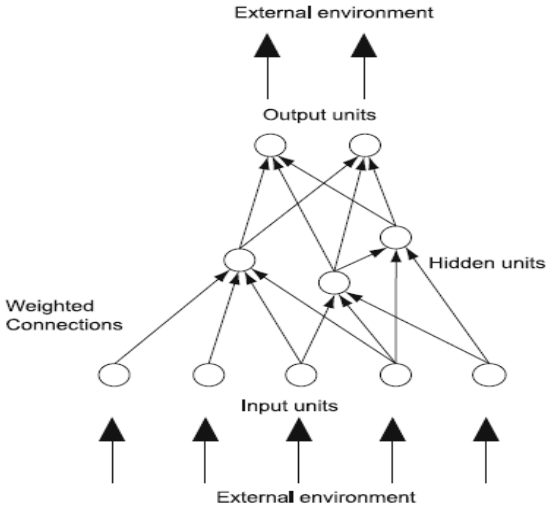

Figure 2. A generic neural network architecture. It consists of input and output units which are connected to the external environment and hidden units which are only connected to other neurons. Typically ANN only uses 1 hidden layer.

Designing an ANN involves finding the optimal weight connection values as well as the optimal topology of the network so the correct output can be produced for any given input patterns. For complex tasks with high dimensional search space, machine learning methods have been helpful in alleviating the design

complexity of ANN. Earlier research focused on using supervised approach to fine tune the connection weights for tasks such as speech and handwriting recognition [16].

For robotic systems involving multiple agents, supervised approach has not been applied successfully as providing output behaviour for every single interaction between agents and environments became too complex. *Neuro-evolution* (NE) can be applied in multi-agent systems as it provides unsupervised approach to designing robotic controllers. NE uses evolutionary algorithm to adapt the weights and possibly topology of the network [29]. Many different forms of NE have been developed which has its own advantages and limitations. We will be only focusing on two NE methods NEAT and SANE which will be discussed further in section 2.

1.5 Evolutionary Robotics

Good example of modern controller design method can be found in *evolutionary robotics* [27]. Evolutionary robotics (ER) is a novel discipline that applies evolutionary algorithms mentioned in section 1.2 to robot design and control. This typically involves evolving an ANN controller with neuro evolution. ER bypasses the design problem faced by the traditional controller design approach, as it does not require any form of decomposition of global behaviour, instead the control system is evaluated as a whole, only looking at the emergent global behaviour. Controller developed with ER has shown fast adaptations [19]. Successful Application of ER in controller design includes obstacle avoidance and robot soccer [6].

Main focus of this paper will be on the controller design in evolutionary robotics. Specifically this paper will empirically evaluate the performance of NEAT and SANE as an evolutionary controller design method for a multi-agent system. First aim of this research is to figure out which of the two methods are superior at controller design for multi-agent system. Secondary aim is to potentially discover which qualities of neuro-evolution method leads to it being a good or a bad controller designer. These methods will be tasked with collective gathering tasks, where they need to train a team of robots to cooperate in collecting various objects scattered in the environment. We hypothesize that NEAT's ability to produce minimal solution will allow it to produce good solutions faster, but in the long run genetic diversity of SANE will allow it to produce better solutions.

2. METHODS

2.1 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) [26] is a NE technique that follows a constructive approach, with initial population starting with networks starts with minimal structure (zero hidden nodes). The topology of the network is evolved incrementally by adding additional hidden nodes and connections through mutation and crossover. This process is referred to as *complexification* and it reduces the search space dimensions drastically, allowing solutions to be found faster.

NEAT's genotype consists of node genes representing the nodes of the network that can be connected, and connection genes each of which specifies the connections between the two node genes. Connection genes are marked with what is known as an innovation number that indicates the historical origin of the connection. Whenever a new gene emerges through mutation (which indicates new topology has emerged), the global *innovation number* count is incremented and is assigned to that new gene. Once assigned, an innovation number of a gene is never changed, even when it is passed off to an offspring through recombination. If somehow through mutation same structural innovation occurs multiple times, these genes are assigned to the same innovation number. NEAT uses this information to avoid *competing convention* problem, which causes low fitness in offspring due to inheriting duplicate structure from its parents [29].

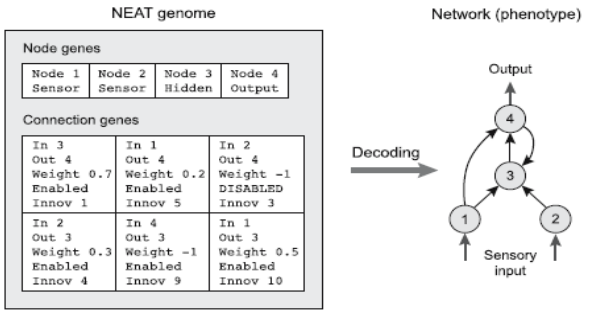

Figure 3. Genetic encoding of a network in NEAT. Genetic operators can either insert new genes or disable old genes.

Mutation operators in NEAT can modify both the connection weights and the structure of the network. Weight values are mutated as any other NE methods, where each connection has a chance to be perturbed to new value. Structure can be mutated in two ways. Frist structure mutation adds new connection between existing nodes. Second structure mutation adds new node by disabling an existing connection and inserting a node where the connection used to be. New node is then connected to the two nodes that used to be joined by the old connection.

NEAT uses the innovation number of genes to perform crossover between two genotypes. Since each structure is associated with a unique innovation number, genes with same innovation number must represent same structure. NEAT avoid duplicate structures from being passed on to the offspring by lining up the genes with same innovation number (matching genes). The offspring's genome is created by taking the matching gene randomly from either parent and selecting non-matching genes from the more fit parent,

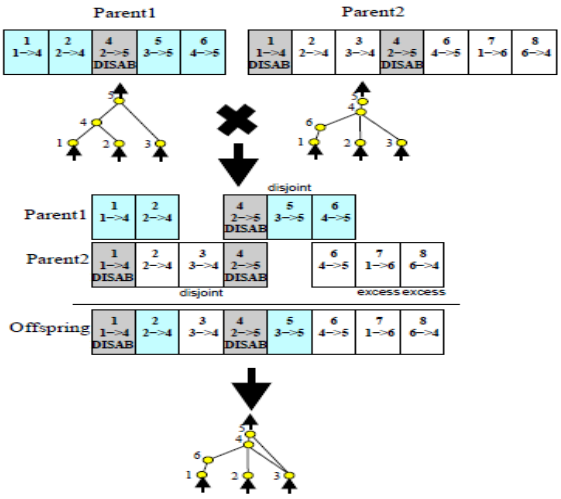

Figure 4. During crossover innovation number of a gene is used to line up the genomes of the two parents. Matching genes are inherited from either parent at random. In this case equal fitness is assumed, so the disjoint (not matching in the middle) and excess genes (not matching in the end) are also inherited randomly.

Speciation or *niching* technique is used as a mechanism to protect new innovations in ANN topology as well as preserving genetic diversity. The population is divided into multiple sub-groups called species based on their degree of similarity which are measured with compatibility distance function. It uses factors such as the number of excess (E) genes and disjoint (D) genes as well as the average weight difference of matching genes (W) in an equation as follows:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W} \qquad \textbf{(3)}$$

N is the size of the larger genome to normalize the genome size, while c1, c2 and c3 are the coefficients to allow adjusting the relative importance of the three factors. In each generation a genome is compared with a representative genome from each species using equation (3). If the value is less than the threshold compatibility value the genome will be put into that species. If a genome fails to match with any species, new species will be created for it. This gives new species some time to optimize before it starts competing with other species.

Explicit fitness sharing is implemented so that individuals in the same speciation group share their fitness. This effectively penalizes species that becomes dominant which prevents one species from taking over the whole population. It also allows any new innovative topology that do not fall under any existing species a chance to adapt its weights in a new speciation group before competing with the rest of population.

When tested on benchmark tests such as pole balancing and double pole balancing tests, NEAT outperformed other neuro-evolution approaches including SANE [24].

2.2 SANE

Symbiotic, adaptive neuro-evolution (SANE) [18] is another NE technique for evolving ANN controllers. Most NE methods evolve complete solution in a form of neural network which is evaluated independently to the other individuals in the population. This approach may direct the search toward the best individual which may lead to premature convergence preventing discovering potentially better solutions. SANE differs from these approaches by evolving partial solutions that depend on other member of the population for evaluation. Each individual in SANE represents a hidden node of the network and they are evaluated by measuring how well they combine with other neurons to form the hidden layer of the network. Complete network are formed by randomly combining multiple neuron individuals into a hidden layer to create a complete network with predefined input and output layer.

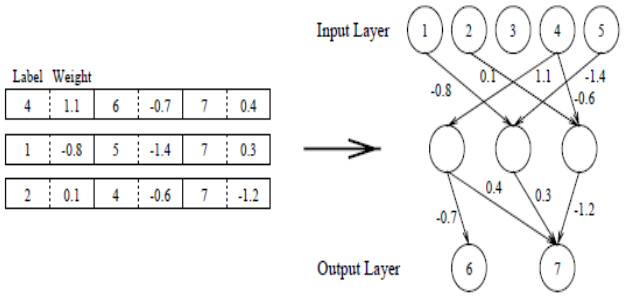

Figure 5. A three layer network is created from 3 neuron definitions. The neurons are shown on the left while the corresponding network is shown on the right.

The main benefit of SANE is that it preserves genetic diversity. Neurons are required to cooperate with each other to complete the given task. Unless the task is extremely simple, no single neuron should be able to perform the task by itself. This requires the neuron population to contain genetically diverse individuals whom provides different functionalities to form an effective network. Another advantage of SANE is that it is able to identify good building blocks of s neural network. In traditional approach each neuron in a network is only used for the network that it is currently in. This implies that good neurons in a bad network cannot be recognized as it is overshadowed by the negative contribution of its neighbouring neurons. SANE can test neurons in different networks allowing for more accurate evaluation of their contribution.

Although the explorative nature of SANE works well in simple benchmarks, not being able to fully exploit the good neuron combinations made it ineffective for a more difficult task requiring high-precision within the solution space. To address this issue, modern variation of SANE performs a hierarchical evolution by evolving additional blueprint population along with the neuron population [17]. Blueprint population keeps track of the good hidden neuron formations found in previous generations. In future generation the hidden layer is formed based on these blueprints. This approach allows exploitation of good neuron combinations to help direct the search in a promising area.

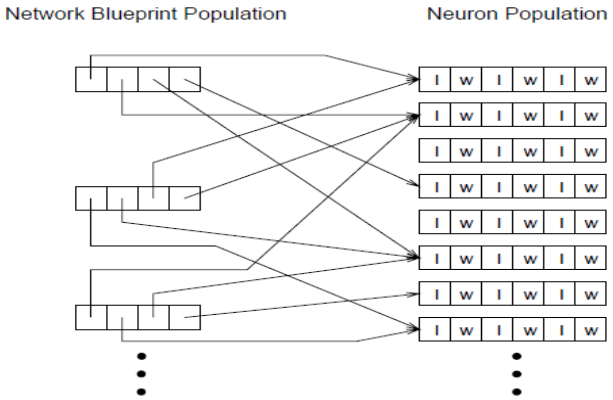

Figure 6. An overview of the relationship between the blueprint population and the neuron population. Each neuron individual specifies the connection to be made within the population. Each blueprint contains pointers to neurons to include in the network.

Genotype chromosome for neuron individuals consists of series of connection definitions. Each definition contains a label field defining the node to connect to, and the weight field defining the value of the connection. Connections can only be made to the nodes in input or output layer. Genotype for the blueprint individuals consists of series of pointers to neurons which are used to create the hidden layer of the network in the evaluation phase.

At the end of the evaluation phase each blueprint receives a fitness score depending on how well its network has performed while the neurons receives average fitness of the best five networks it has participated in. During reproduction phase each neuron individuals in top 25% are mated with each other to create two new offspring. One of the offspring is created through 1-point crossover while the other is the copy of one of the parents. Copying one of the parents as offspring can reduce the harmful effects of adverse mutation which participate in many networks.

These offspring replaces the bottom 50% of the population. As the final step 10% mutation rate per chromosome position is applied to the whole neuron population. Blueprint reproduction follows the elite breeding strategy similar to the neuron reproduction, creating two offspring each by mating the top 25% individuals using 1-point crossover. There are two mutation operator used for blueprint population. First operator replaces the neuron pointer of the blueprint to another random neuron at a rate of 10% per pointer. Second operator replaces the pointer to breeding neuron with pointer to one of its offspring at a rate of 50% per pointer. These selective mutations allow new and unexplored neurons to participate in the network to help maintain diversity. Like neuron population, the blueprint offspring also replaces the bottom 50% of the population

$$
\begin{aligned}
&\text{for each neuron } n \text{ in population } P_n \\
&\quad n.fitness \leftarrow 0 \\
&\quad n.participation \leftarrow 0 \\
&\text{for each blueprint } b \text{ in population } P_b \\
&\quad nn \leftarrow \mathbf{decode}(b) \\
&\quad b.fitness \leftarrow \mathbf{task}(nn) \\
&\quad \text{for each } n \text{ in } b \\
&\qquad n.fitness \leftarrow n.fitness + b.fitness \\
&\qquad n.participation \leftarrow n.participation + 1 \\
&\text{for each neuron } n \text{ in population } P_n \\
&\quad n.fitness \leftarrow n.fitness \ / \ n.participation
\end{aligned}
$$

Figure 7. An overview of the evaluation phase of SANE.

SANE has been applied to robotics domain where it managed to produce high quality neuro-controller for a Khepera robot [22].

3. IMPLEMENTATION

3.1 Collective Gathering Task

The task to be used in our experiment is the collective gathering task. This task requires team of robots to roam around the area to find and retrieve target objects back to the home area. This task can be setup with variety of configurations such as presence of obstacles, multiple destinations, cooperation requirement etc. In this experiment we will be adding an element of cooperation by having certain objects only be movable when pushed by multiple robots. We have also added a slight obstacle avoidance by placing trash objects that needs to be avoided. We will be conducting tests in three different environment configurations with varying difficulties to see how the two methods perform at adapting for more complex tasks.

3.2 Simulation Environment

To test our experiments we have created a virtual environment to simulate the collective gathering task. The environment is an extension of the work done by Hewland [12] which uses Jbox2D and MASON library to create a 2D environment for robot team simulation. The environment will consist of team of robots and a number of collectable objects with different dimensions that either needs to be collected or avoided. The environment is separated into two areas. The first area is the foraging region where objects will be randomly dispersed. Second area is called nest or home region where the collected objects need to be deposited.

3.3 Collectable Objects

Collectable objects come in three sizes – small, medium and large. Small objects can be pushed by a single robot whereas medium and large robot requires two and three robots to move respectively. This promotes collective cooperative behaviour to emerge as robots need to work together to collect larger objects.

Objects are further classified as either a resource or trash. Collecting Resource objects add positively to the team's fitness while collecting trash objects add negatively to the team's fitness. Thus team should develop behaviour to collect as many resources as possible while trying avoiding as much trash objects.

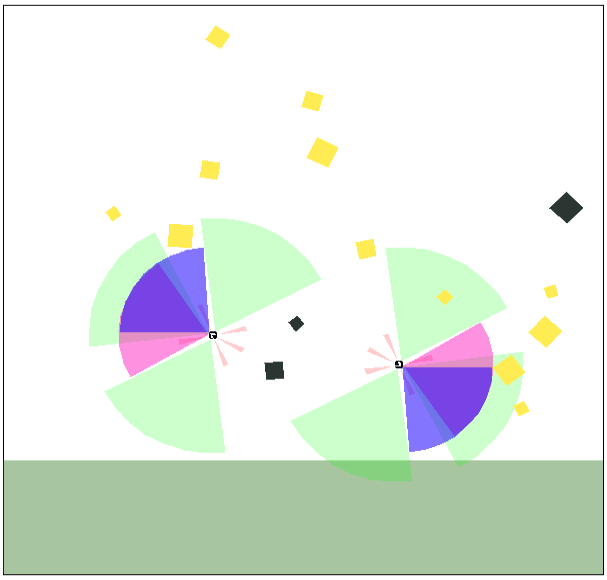

Figure 8. Graphical view of simulated environment. Home region is highlighted at the bottom. Resource objects are shown in yellow while trash objects are shown in black. Two agents along with their sensors' range and FoV are also shown.

3.4 Collectable Objects

Collectable objects come in three sizes – small, medium and large. Small objects can be pushed by a single robot whereas medium and large robot requires two and three robots to move respectively. This promotes collective cooperative behaviour to emerge as robots need to work together to collect larger objects.

Objects are further classified as either a resource or trash. Collecting Resource objects add positively to the team's fitness while collecting trash objects add negatively to the team's fitness. Thus team should develop behaviour to collect as many resources as possible while trying avoiding as much trash objects.

Table 1. Table showing the composition of collectable objects (resource/trash) for three environment configuration.

	Small	*Medium*	*Large*
Environment 1	8/2	4/1	0/0
Environment 2	4/1	4/1	4/1
Environment 3	0/0	4/1	8/2

3.5 Robot Teams

Simulated robot agents in our experiment were designed to be similar to the Khepera robot [12]. Our robot agent has a circular shaped body with sensors attached at various positions around the body. There are two wheels attached at the bottom of the robot which supports backward and forward movements. The controller

determines the behaviour of the object by modifying the velocity of these two wheels. To simplify our experiments, homogenous team composition was used, in which each agent in a team has a copy of the same controller and sensory configuration.

3.5.1 Sensors

Each robot agents will be equipped with multiple sensors to acquire accurate information about the surrounding environment. We have designed four types of sensors with each sensor having its own advantages and limitations. (1) Infrared and (2) Ultrasonic sensors returns the distance to the closest sensed object. Infrared sensors have a relatively short range. Ultrasonic sensors have much longer range and field of view but are unable to sense objects that are very close. (3) Colour sensor is used to determine the type of the closest objects, which allow the robots to detect and avoid unwanted objects. (4) Low-resolution camera sensor was designed to enable robots to recognize situations requiring cooperation. It does so by scanning an area and returning the ratio that measures density of robots to the density of collectable objects. Every sensor's reading is normalized to be between 0 and 1.

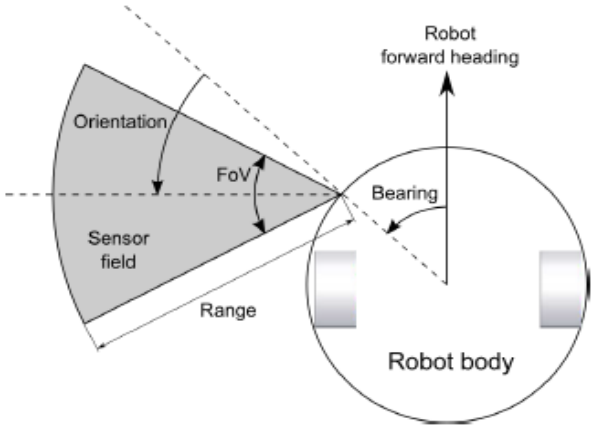

Figure 9. Overview of sensor implementation. Bearing determines the position of the sensor relative to the robot's forward heading. Orientation determines the direction the sensor is facing relative to its bearing. Range and field of view (FoV) determines how long and wide the sensors can see.

Morphology for each agent will consist of 5 infrared sensors, 3 ultrasonic sensors and 1 of each colour and low-res camera sensor. Detailed specifications of these sensors are outlined in table 2. In addition to these sensor there is an additional sensor attached to the bottom of the robot which distinguishes the area below the robot as either the foraging region or the home region. In total each agent will be equipped with 11 sensors, which provided input to robot's neuro controller.

Table 2. Morphology implementation details

	Infrared	*Ultrasonic*	*Colour*	*Camera*
Quantity	5	3	1	1
Bearing	±40°,±140°,180°	0°,±90°	15°	-15°
Orientation	0	0	0	0
FoV	0.2	1.22	1.5	1.5
Range	0 ~ 1.0	0.2 ~ 4.0	0 ~ 3.0	0 ~ 3.0

3.5.2 Controllers

Both methods evolved a population of 100 neural network individuals for 250 generations. Each ANN had 11 input nodes corresponding to the 11 readings received from its sensors. Size of the hidden layer differed for the two methods. Output layer consisted of 2 nodes. The final output of ANN determined the final speed and direction of the robot's movement. Each node in hidden layer and output layer used sigmoid transfer function. Each connection was normalized to be between -1 and 1 at all times.

3.5.2.1 NEAT

We have borrowed an implementation of NEAT from Hewland's framework, which used a java machine learning library Encog [12]. NEAT's population consisted of 100 NEAT networks. Each member of the population was initialized with zero hidden nodes with initial connection density of 50% of maximum possible number of connections. Initial value of these connections was randomized. Once initialized the controller was subject to the NEAT's adaptation process. NEAT was able to adapt the number of hidden nodes using crossover, link mutation and node mutation operators. For each generation Encog repeatedly selects one of these operators to generate offspring until enough offspring have been generated. Crossover operator uses truncation selection (selects random from top *x*%) to select two parents and produce one new offspring. Mutation operators creates a copy of the parent and generate an offspring by mutating the copy. NEAT used 30% elite rate, preserving the top 30% of the population for the next generation.

Table 3. NEAT parameters

Parameters	*Values*
Population size	100
Truncation selection	Top 30% of population
Elitism	30%
Speciation constants	c1= 1.0, c2=1.0, c3=0.4
Threshold compatibility	3.0
Initial connection density	50%
Connection weight range	[-1.0:1.0]
Crossover probability	50%
Link weight probability	49.3%
Add node probability	0.1%
Add link probability	0.5%
Remove link probability	0.1%

3.5.2.2 SANE

SANE's neuron population consisted of 800 neurons. Each neuron individual had a total of 9 connections which they could use to form connection with any of the 13 input and output nodes. During initialization each neuron's connections as well as its values are assigned randomly, not allowing duplicate connections. 1-point crossover was used for neuron crossover with point of intersection chosen at random. Only the top 25% of the population are allowed to reproduce. Uniform mutation was used for neuron population with a rate of 10% per chromosome position. Elitism of 50% was used to preserve the top 50% of the population while replacing the bottom portion with new individuals.

SANE's blueprint population consisted of 100 blueprints. Each blueprint contained 8 pointers to individuals in neuron population.

These pointers are randomly initialized.. Similarly to neuron population, 1-point crossover was used on top 25% of the population to create new blueprint individuals. For mutations, rate of 10% was used for tier-1 mutation that switched a pointer in a blueprint to a random neuron. 50% mutation rate was used for tier-1 mutation that switched pointer to a neuron to its offspring. Like neuron population 50% elite rate was used.

Table 4. SANE parameters

Parameters	*Values*
Population size	800/100
Truncation selection	Top 25% of population
Elite rate	50%
Connection per hidden neuron	9
Connection weight range	[-1.0,1.0]
Neuron mutation rate	10%
Blueprint tier-1 mutation rate	10%
Blueprint tier-2 mutation rate	50%

3.5.3 Heuristics

To reduce complexity of the task and speed up the experiment, we have added some heuristic behaviour to the robots to help them with their mission. If a robot moves within a gripping distance of a collectable object, it would attach itself to the object and attempt to move in the direction of the home region. If it cannot move the object it waits for other robots to cooperate for a short duration before detaching itself to find a different object. When robot successfully manages to bring an object to a home-region it immediately detaches itself and searches for other objects.

4. RESULTS

We have tested the performance of the two neuro-evolution methods NEAT and SANE on a collective foraging task on a simulated environment. Each method was tested on three different environment setups (table 1), resulting in total of six tests. Each test was run 20 times and the results are shown below (figure 11).

For environment 1 which required relatively small degree of cooperation to complete (mainly consisting of small objects), NEAT produced its best solution at generation 98 which achieved 94% of maximum performance efficiency. For the same environment SANE produced its best solution at generation 244 that achieved 61% of maximum performance efficiency.

For environment 2 which required slightly more cooperation between robots (all block sizes evenly distributed), NEAT produced its best solution at generation 191 that achieved 76% of the maximum performance efficiency. For the same environment SANE produced its best solution at generation 179 that achieved 44% of maximum performance efficiency.

For environment 3 which required which required high degree of cooperation between robots to solve, NEAT produced its best solution at generation 143 that achieved 58% of the maximum performance efficiency. For the same environment SANE produced its best solution at generation 63 that achieved 13% of maximum performance efficiency.

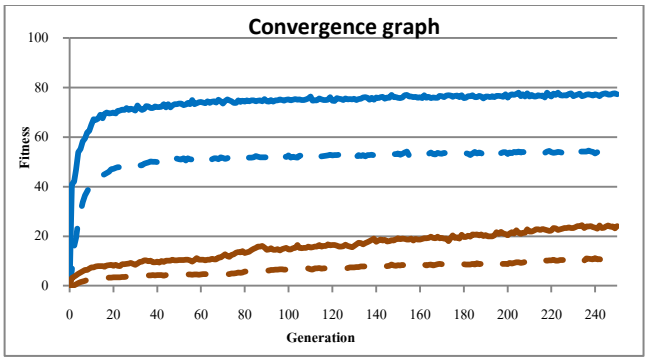

Figure 10. Convergence graph for environment 1. NEAT methods are shown in blue and SANE methods are shown as brown. Solid lines are the best fitness produced in each generation while dotted lines are the average fitness produced in each generation. Similar patterns were observed for other environments.

For all environments, general behaviour of the fittest robot team trained with SANE involved moving in a small circular trajectory around its starting position until an object is detected by the colour ranged sensor, then deciding to either move towards or away from it depending on the type of the sensed object. The fittest team trained with NEAT behaved similarly but they moved in a straight line in a direction they are facing, turning left sharply whenever a wall was encountered. This navigation behaviour allowed them to cover larger portion of the environment leading to faster discovery of collectable objects. Also team trained with NEAT were able to classify objects more accurately, allowing them to avoid trash objects more accurately compared to the team trained with SANE.

The results indicate that NEAT is the more effective approach than SANE on controller design for a collective foraging task. Best solution produced by NEAT was higher than the one produced by SANE on all three environments. Previous research has shown similar results for benchmark controller problems [24], and this result supplements those research by indicating the same holds true for multi-agent collective behaviour tasks.

The Boxplot graphs also provide insights regarding the two methods' consistency. The results from SANE method were positively skewed with big gaps between median and the maximum value. In contrast NEAT produced symmetric data patterns with short range between median and the maximum value. This indicates that NEAT produced good results more consistently, whereas for SANE only a small portion of the runs were able to produce decent results.

It was observed that most tests run with SANE that had a poor start took extremely long before seeing any noticeable fitness improvements. One theory to explain such issue is poor initialization of neuron population. SANE strongly depends on the quality of its neurons to form effective networks. If the initial neuron population is filled with bad individuals, or does not represent the search space correctly by having the population concentrated on a small portion of the search space, network formed from such neuron population will lead to low performance. This issue is compounded by the fact that SANE employs a very conservative mutation operator on neuron population, which makes finding new neurons difficult. It seems although SANE is good at maintaining genetic diversity, it struggles to introduce new genetic material to its population once it loses its diversity.

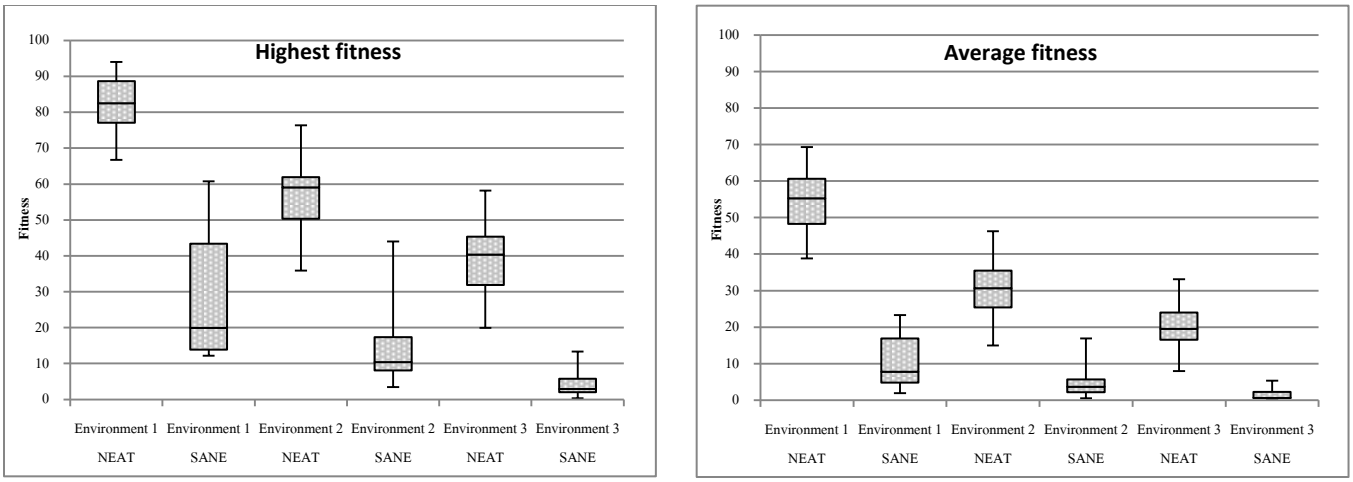

Figure 11. Boxplots of the maximum fitness team evolved in each experiment are shown on the left. Right shows boxplots of the average fitness of the population after 250 generations. All tests were run 20 times each.

Apart from the performance, our results also provide information regarding the efficiency of the two methods. From the convergence graph in fig 10 it can be seen that NEAT converges rapidly, finding a local optimal solution in a short amount time. On the other hand SANE has a very slow convergence rate and it still does not converge fully at generation 250. Although premature convergence prevents a method from discovering potential better solutions, not being able to find any reasonable local solution within a given timeframe makes the method impractical. This experiment demonstrated that for a complex controller tasks, SANE's explorative approach to maintaining genetic diversity can hinder the fitness growth rate of the population as it does not fully exploit the local solutions.

NEAT's superior efficiency in finding optimal solution can be theorized as the result of its complexification scheme. NEAT's constructive approach to start with no hidden layer allows it to search through minimal number of weight dimensions, leading to fast discovery of promising locations to be explored further. In addition, high dimensional structures in NEAT are derived by elaborating on lower dimensional structures that have been optimized already in previous generations. This allows the search process in high dimensional space to start in a promising area, whereas for fixed topology NE methods such as SANE it would start on a random location. For this reason it has been argued that complexification evolution can find solutions that are unlikely to be found by fixed-topology evolution such as SANE [25].

Future work will revolve around experimenting with different evolution parameters such as increasing the number of generations to see the long term performances of the methods. Also, Implementing and testing more neuro-evolution methods will allow more accurate performance evaluations of each method. One method to be implemented is ESP [11] which is an extension of SANE that showed promising results in traditional benchmark tests [24]. Another extension would be to use a heterogeneous team composition which allows each robot to have different behavioural controllers to its teammates. Heterogeneous composition allows individual robots to take on a specialized role which could lead to team's performance as a whole [20]. For example for the collective gathering tasks, certain robots can take on a role of a scouter that informs other robots of locations of the resources, or a detector which verifies for their team whether a certain object should be avoided or collected. In addition the simulation environment could be modified to test different robotic tasks. For example this paper focused on the collective gathering task, but with addition of obstacles and modification to the composition of the collectable objects, the task can be focused towards collision avoidance problem.

5. CONCLUSION

Neuro-evolution provides a valuable way of developing a controller for evolutionary robotics systems. We have evaluated the performance of the two popular neuro evolution methods NEAT and SANE on a collective gathering task. From our results it was demonstrated that NEAT outperformed SANE by a significant margin. Reason for such outstanding performance by NEAT can be credited to the complexification scheme which reduces search space greatly leading to quick discovery of good solutions. SANE's main advantage of keeping genetic diversity may be a double edged sword as it seemed slows down the fitness growth rate significantly.

However this research alone is not enough to make definite conclusions regarding the two methods. There are many areas of improvements for this experiment such as implementing more methods to compare, parameter optimization which suffered due to the time and resource constraints. Further research will be required to verify the results demonstrated in this paper.

6. ACKNOWLEDGMENTS

Computations were performed using facilities provided by the University of Cape Town's ICTS High Performance Computing team: http://hpc.uct.ac.za

7. REFERENCES

[1] B. D. Argall, S. Chernova, M. Veloso and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems,* vol. 57, pp. 469-483, 2009.

[2] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, 1997, pp. 12-20.

[3] A. G. Barto, *Reinforcement Learning: An Introduction.* MIT press, 1998.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning.* springer New York, 2006.

[5] A. H. Bond and L. Gasser, "An analysis of problems and research in DAI," 1988.

[6] J. Chavas, C. Corne, P. Horvai, J. Kodjabachian and J. Meyer, "Incremental evolution of neural controllers for robust obstacle-avoidance in khepera," in *Evolutionary Robotics*, 1998, pp. 227-247.

[7] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Springer Science & Business Media, 2003.

[8] A. Engelbrecht, *Computational Intelligence : An Introduction.* South Africa: WILEY, 2007.

[9] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies.* MIT press, 2008.

[10] L. J. Fogel, "Autonomous automata," *Industrial Research*, vol. 4, pp. 14-19, 1962.

[11] F. J. Gomez and R. Miikkulainen, "Solving non-markovian control tasks with neuroevolution," in *IJCAI*, 1999, pp. 1356-1361.

[12] J. Hewland and G. Nitschke, "Adaptive Robot Team Behavior and Morphology for Collective Gathering," 2015.

[13] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* U Michigan Press, 1975.

[14] S. B. Kotsiantis, I. Zaharakis and P. Pintelas, *Supervised Machine Learning: A Review of Classification Techniques*, 2007.

[15] J. R. Koza, *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Stanford University, Department of Computer Science, 1990.

[16] T. Mitchell, *Machine Learning.* New York; London: McGraw-Hill, 1997.

[17] D. E. Moriarty and R. Miikkulainen, "Hierarchical evolution of neural networks," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., the 1998 IEEE International Conference On*, 1998, pp. 428-433.

[18] D. E. Moriarty and R. Mikkulainen, "Efficient reinforcement learning through symbiotic evolution," *Mach. Learning*, vol. 22, pp. 11-32, 1996. [19] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT press, 2000.

[20] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Auton. Agents Multi-Agent Syst.*, vol. 11, pp. 387-434, 2005.

[21] I. Rechenberg, "Cybernetic solution path of an experimental problem," 1965.

[22] N. Richards, D. E. Moriarty and R. Miikkulainen, "Evolving neural networks to play Go," *Appl. Intell.*, vol. 8, pp. 85-96, 1998. [23] S. Russell, P. Norvig and A. Intelligence, "A modern approach," *Artificial Intelligence.Prentice-Hall, Egnlewood Cliffs*, vol. 25, 1995.

[24] M. Shi, "An empirical comparison of evolution and coevolution for designing artificial neural network game players," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, 2008, pp. 379-386.

[25] K. O. Stanley and R. Miikkulainen, "Competitive coevolution through evolutionary complexification," *J.Artif.Intell.Res.(JAIR)*, vol. 21, pp. 63-100, 2004.

[26] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99-127, 2002.

[27] V. Trianni, *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots.* Springer Science & Business Media, 2008.

[28] L. Wang, K. C. Tan and C. M. Chew, *Evolutionary Robotics: From Algorithms to Implementations.* World Scientific, 2006.

[29] X. Yao, "Evolving artificial neural networks," *Proc IEEE*, vol. 87, pp. 1423-1447,