14

A Quantitative Analysis of Memory Usage for Agent Tasks

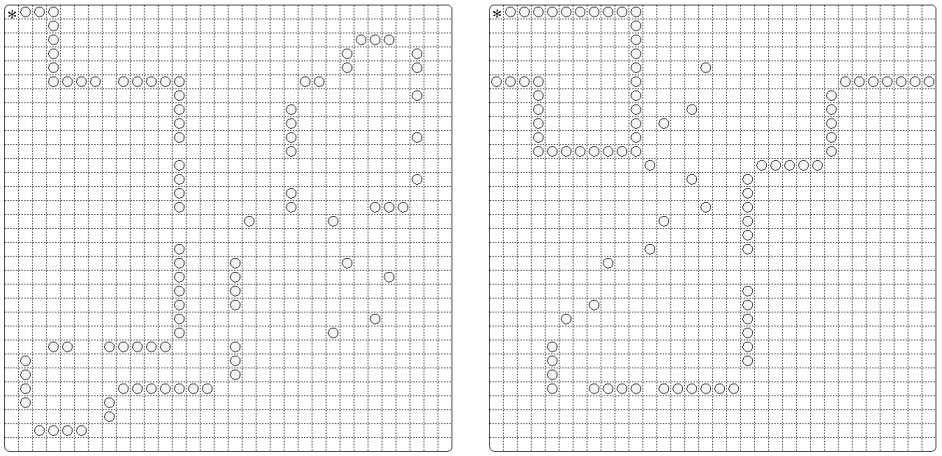
DaeEun Kim
Yonsei University, School of Electrical and Electronic Engineering
Corea (South Korea)

1. Introduction

Many agent problems in a grid world have been handled to understand agent behaviours in the real world or pursue characteristics of desirable controllers. Normally grid world problems have a set of restricted sensory configurations and motor actions. Memory control architecture is often needed to process the agent behaviours appropriately. Finite state machines and recurrent neural networks were used in the artificial ant problems (Jefferson et al., 1991). Koza (1992) applied genetic programming with a command sequence function to the artificial ant problem. Teller (1994) tested a Tartarus problem by using an indexed memory. Wilson (1994) used a new type of memory-based classifier system for a grid world problem, the Woods problem.

The artificial ant problem is a simple navigation task that imitates ant trail following. In this problem, an agent must follow irregular food trails in the grid world to imitate an ant's foraging behaviour. The trails have a series of turns, gaps, and jumps on the grid and ant agents have one sensor in the front to detect food. Agents have restricted information of the surrounding environment. Yet they are supposed to collect all the food on the trails. The first work, by Jefferson et al. (1991), used the John Muir trail, and another trail, called Santa Fe trail, was studied with genetic programming by Koza (1992). The trails are shown in Fig. 1. This problem was first solved with a genetic algorithm by Jefferson et al. (1991) to test the representation problem of controllers. A large population of artificial ants (65,536) were simulated in the John Muir trail with two different controller schemes, finite state machines and recurrent neural networks. In the John Muir trail, the grid cells on the left edge are wound adjacent to those on the right edge, and the cells on the bottom edge are wound adjacent to those at top. Each ant has a detector to sense the environment and an effector to wander about the environment; one bit of sensory input to detect food and two bits of motor actions to move forward, turn right, turn left and think (no operation). Its fitness was measured by the amount of food it collects in 200 time steps. At each time step, the agent can sense the environment and decide on one of the motor actions. The behaviours of ant agents in the initial population were random walks. Gradually, more food trails were traced by evolved ants. Koza (1992) applied genetic programming to the artificial ant problem with the Santa Fe trail (see Fig. 1(b)), where he assumed it as a slightly more difficult trail than the John Muir trail, because the Santa Fe trail has more gaps and turns between food pellets. In his approach, the control program has a form of S-expression (LISP) including a sequence of actions and conditional statements.

Source: Frontiers in Evolutionary Robotics, Book edited by: Hitoshi Iba, ISBN 978-3-902613-19-6, pp. 596, April 2008, I-Tech Education and Publishing, Vienna, Austria



(a) (b)

Figure 1. Artificial ant trails (*: ant agent, circle: food) (a) John Muir trail (Jefferson et al., 1991) (b) Santa Fe trail (Koza, 1992)

Many evolutionary approaches related to the artificial ant problem considered the fitness as the amount of food that the ant has eaten within a fixed number of time steps (Jefferson et al.,1991, Koza,1992, Angeline,1998, Silva et al.,1999). The approaches suggested a new design of control structures to solve the problem, and showed the best performance that they can achieve. The artificial ant problem is an agent problem that needs internal memory to record the past sensory readings or motor actions. However, there has been little discussion for the intrinsic properties related to memory to solve the problem, although the control structures studied so far have a representation of internal memory.

Internal memory is an essential component in agent problems in a non-Markovian environment (McCallum, 1996, Lanzi, 1998, Kim and Hallam, 2002). Agents often experience a perceptual aliasing problem (when the environmental features are not immediately observable or only partial information about the environment is available to an agent, the agent needs different actions with the same perceived situation) in non-Markovian environment. For instance, in the artificial ant problem an ant agent has two sensor states with one sensor, food detected or not in the front, but it needs different motor actions on the same sensor state, depending on the environmental feature. Thus, a memoryless reactive approach is not a feasible solution for the problem. There have been memory-encoding approaches to solve agent problems or robotic tasks in a non-Markovian environment or partially observable Markov decision process (POMDP). Colombetti and Dorigo (1994) used a classifier system to learn proper sequences of subtasks by maintaining internal state and transition signals which prompt an agent to switch from one subtask to another. Lanzi (2000) has shown that internal memory can be used by adaptive agents with reinforcement learning, when perceptions are aliased. Also there have been researches using a finite-size window of current and past observations and actions (McCallumn, 1996, Lin and Mitechell, 1992). Stochastic approaches or reinforcement learning with finite state controllers have been applied to POMDPs (Meuleau et al., 1999, Peshkin et al., 1999, Braziunas and Boutiler, 2004). Bram and de Jong (2000) proposed a means of counting the number of internal states required to perform a particular task in an environment. They estimated state counts from

finite state machine controllers to measure the complexity of agents and environments. They initially trained Elman networks (Elman, 1990) by reinforcement learning and then extracted finite state automata from the recurrent neural networks. As an alternative memory-based controller, a rule-based state machine was applied to robotic tasks to see the memory effect (Kim and Hallam, 2001). Later Kim and Hallam (2002) suggested an evolutionary multiobjective optimization method over finite state machines to estimate the amount of memory needed for a goal-search problem.

Generally, finding an optimal memory encoding with the best behaviour performance in non-Markovian environments is not a trivial problem. Evolutionary computation has been a popular approach to design desirable controllers in agent problems or robotic tasks (Nolfi and Floreano, 2000). To solve the artificial ant problem, we will follow the evolutionary robotics approach. In the evolutionary approach, the behaviour performance of an ant agent is scored as fitness and then the evolutionary search algorithm with crossover and mutation operators tries to find the best control mapping from sensor readings to actions with a given memory structure. Here, we focus on the question of how many memory states are required to solve the artificial ant problem in non-Markovian environment or what is the best performance with each level of memory amount. This issue will be addressed with a statistical analysis of fitness distribution.

An interesting topic in evolutionary computation is to estimate the computational effort (computing time) to achieve a desirable level of performance. Koza (2002) showed a statistic to estimate the amount of computational effort required to solve a given problem with 99% probability. In his equation, the computational effort $I(m,z)$ is estimated as

$$I(m,z) = m\, g\, \log(1-z) / \log(1-P(m,g)) \quad (1)$$

where m is the population size, g is the number of generations, z is the confidence level which is often set to 0.99, and $P(m,g)$ is the probability of finding a success within mg evaluations. However, this equation requires to estimate $P(m,g)$ accurately. It has been reported that the measured computational effort has much deviation from the theoretical effort (Christensen and Oppacher, 2002, Niehaus and Banzhaf, 2003). The probability $P(m,g)$ has been measured as the number of successful runs over the total number of runs. In that case, $P(m,g)$ does not consider how many trial runs are executed, which can lead to the inaccurate estimation of the computational effort. The estimation error can be observed, especially when only a small number of experimental runs are available (Niehaus and Banzhaf, 2003). Lee (1998) introduced another measure of the computational effort, the average computing cost needed to obtain the first success, which can be estimated with $mg(\alpha+\beta+2)/(\alpha+1)$ for given α successes and β failures. As an alternative approach to the performance evaluation, the run-time distribution, which is a curve of success rate depending on computational effort, has been studied to analyze the characteristics of a given stochastic local search (Hoos and Stuetzle, 1998, 1999). However, this measure may also experience the estimation error caused by variance of success probability.

Agent problems in a grid world have been tested with a variety of control structures (Koza, 1992, Balakrishnan and Honavar,1996, Ashlock, 1997,1998, Lanzi, 1998, Silva et al., 1999, Kim, 2004), but there has been little study to compare control structures. Here, we introduce a method of quantitative comparisons among control structures, based on the behaviour performances. The success rate or computational effort is considered for the purpose. In this paper we use finite state machines as a quantifiable memory structure and a varying

number of memory states will be evolved to see the memory effect. To discriminate the performances of a varying number of memory states, we provide a statistical significance analysis over the fitness samples. We will present a rigorous analysis of success rate and computational effort, using a beta distribution model over fitness samples. Ultimately we can find the confidence interval of the measures and thus determine statistical significance of the performance difference between an arbitrary pair of strategies. This analysis will be applied to show the performance difference among a varying number of internal states, or between different control structures. The approach is distinguished from the conventional significance test based on the *t*-statistic. A preliminary report of the work was published in the paper (Kim, 2006).

We first introduce memory-encoding structures including Koza's genetic programming and finite state machines (section 2) and show methods to evaluate fitness samples collected from evolutionary algorithms (section 3). Then we compare two different control structures, Koza's genetic programming controllers and finite state machines, and also investigate how a varying number of internal states influence the behaviour performance in the ant problem. Their performance differences are measured with statistical significance (section 4).

2. Memory-Encoding Structures

In this section we will show several different control structures which can encode internal memory, especially focusing on genetic programming and finite state machines. The two control structures will be tested in the experiments to quantify the amount of memory needed for the ant problem.

2.1 Genetic Programming approach

Koza (1992) introduced a genetic programming approach to solve the artificial ant problem. The control structure follows an S-expression as shown in Fig. 2. The ant problem has one sensor to look around the environment, and the sensor information is encoded in a conditional statement **if-food-ahead**. The statement has two conditional branches depending on whether or not there is a food ahead. The **progn** function connects an unconditional sequence of steps. For instance, the S-expression **(progn2 left move)**directs the artificial ant to turn left and then move forward in sequence, regardless of sensor readings. The **progn** function in the genetic program corresponds to a sequence of states in a finite automaton.

```
(if-food-ahead  (move)
   (progn3         (left)
                   (progn2  (if-food-ahead    (move) (right))
                            (progn2  (right)
                            (progn2  (left)    (right))))
                   (progn2  (if-food-ahead    (move)  (left))
                             move))))
```

Figure 2. Control strategy for Santa Fe trail by Koza's genetic programming (Koza, 1992）; **(if-food-ahead (move) (right))** means that if food is found ahead, move forward, else turn right, and **progn2** or **progn3** defines a sequence of two actions (subtrees) or three actions (subtrees)

In Koza's approach, a fitness measure for evolving controllers was defined as the amount of food ranging from 0 to 89, traversed within a given time limit. An evolved genetic program did not have any numeric coding, but instead a combination of conditional statements and motor actions were represented in an S-expression tree without any explicit state specification. The evaluation of the S-expression is repeated if there is additional time available. Fig. 2 is one of the best control strategies found (Koza, 1992).

Following Koza's genetic programming approach, we will evolve S-expression controllers for the Santa Fe trail in the experiments. Here, we use only two functions, **if-food-ahead** and **progn2** to restrict evolving trees into binary trees, and **progn3** can be built with the primitive function **progn2**. In the evolutionary experiments, the number of terminal nodes in an S-expression tree will be taken as a control parameter. As a result, we can see the effect of a varying number of leaf nodes, that is, a variable-length sequence of motor actions depending on the input condition. Later we will explain how the control parameter is related with the amount of memory that the S-expression tree uses.

2.2 Finite State Machines

A simple model of memory-based systems is a Boolean circuit with flip/flop delay elements. A Boolean circuit network with internal memory is equivalent to a finite state machine (Kohavi, 1970). Its advantage is to model a memory-based system with a well-defined number of states, and allows us to quantify memory elements by counting the number of states. The incorporation of state information helps an agent to behave better, using past information, than a pure reaction to the current sensor inputs. Finite state machines have been used in evolutionary computation to represent state information (Fogel, 1966, Stanley et al., 1995, Miller, 1996).

A Finite State Machine (FSM) can be considered as a type of Mealy machine model (Kohavi, 1970, Hopcroft and Ullman, 1979), so it is defined as $M = (Q, \Sigma, \Delta, \lambda, q_0)$ where q_0 is an initial state, Q is a finite set of states, Σ is a finite set of input values, Δ is a set of multi-valued outputs, δ is a state transition function from $Q \times \Sigma$ to Q, and λ is a mapping from $Q \times \Sigma$ to Δ, where $\lambda(q,a)$ in Δ. $\delta(q,a)$ is defined as the next state for each state q and input value a, and the output action of machine M for the input sequence $a_1, a_2, a_3, \ldots, a_n$, is $\lambda(q_{x0}, a_1)$, $\lambda(q_{x1}, a_2)$, $\lambda(q_{x3}, a_3)$, $\ldots, \lambda(q_{x(n-1)}, a_n)$, where $q_{x0}, q_{x1}, q_{x3}, \ldots, q_{xn}$ is the sequence of states such that $\delta(q_{xk}, a_{k+1}) = q_{x(k+1)}$ for $k=0,..,n-1$.

state	Input 0	Input 1
q0	*q1,L*	*q0,M*
q1	*q2,R*	*q2,M*
q2	*q3,R*	*q3,R*
q3	*q0,M*	*q4,M*
q4	*q0,M*	*q0,M*

a)

state	Input 0	Input 1
q0	*q7,R*	*q6,M*
q1	*q6,N*	*q2,R*
q2	*q5,R*	*q5,M*
q3	*q0,R*	*q1,L*
q4	*q2,L*	*q5,M*
q5	*q6,N*	*q4,M*
q6	*q0,R*	*q6,M*
q7	*q2,R*	*q2,L*

b)

state	Input 0	Input 1
q0	*q5,R*	*q6,N*
q1	*q0,R*	*q0,N*
q2	*q4,M*	*q3,M*
q3	*q6,R*	*q3,M*
q4	*q5,M*	*q0,L*
q5	*q3,M*	*q4,L*
q6	*q1,R*	*q2,M*

c)

Table 1. Finite state machines for Santa Fe trail problem (a) 405 time steps, with 5 states (b) 379 time steps, with 8 states (c) 356 time steps, with 7 states (input 1: food ahead, input 0: no food ahead), output set is *L* (turn left), *R* (turn right), *M* (move forward), *N* (no-operation)

FSM can be used as a quantifiable memory structure, but developing an optimal state transition mapping needs an exhaustive search and so we apply a genetic algorithm to find desirable FSM controllers for the agent problem. An FSM controller is denoted as a numeric chromosome consisting of integer strings, unlike Koza's genetic programming. The chromosome represents a state transition table in a canonical order and its initial state is 0 by default. That is, the gene coding is defined here as a sequence of the pair (state number, state output) of each sensor value in canonical order of state number. For example, the gene coding in Table 1(b) is represented as:

7R6M 6N2R 5R5M 0R1L 2L5M 6N4M 0R6M 2R2L

where motor actions will also be coded numerically. A set of sensor configurations is defined for each internal state, following the Mealy machine notation. The encoding of the Mealy machine can easily represent sequential states. However, it needs an encoding of complete sensory configurations for each state and scales badly with growing machine complexity. This control structure is useful to agents with a small number of sensors, since the chromosome size in finite state machines is exponentially proportional to the number of sensors. The artificial ant has one sensor, and the Mealy machine will have a reasonable size of chromosome even for a large number of internal states.

If there is a repeated sequence of actions, (A_1, A_2, A_3, …, A_n) to be executed, it can be run by FSM controllers with at most n internal states. Koza's genetic program has a sequence of conditioned actions and so it can be converted into a finite state automaton without difficulty. Terminal nodes in a genetic program of S-expression define motor actions of an ant agent, and the tree traversal by a depth-first-search algorithm relying on sensor readings will guide a sequence of actions. We can simply assign each action in sequence to a separate internal state, and the sequence order will specify the state transition. For example, the function **progn2** or **progn3** has a series of actions, and so the corresponding states defined for the actions will have unconditional, sequential state transitions. The function **if-food-ahead** will have a single internal state for its two actions (the function **if-food-ahead** is supposed to have two children nodes or two subtrees. If an action is not observed immediately at the left or right child, the first terminal node accessed by the tree traversal will be chosen for the internal state.) in the branches, because the actions depending on the sensor reading (input 0 or 1) can be put together in a slot of internal state. With this procedure, we can estimate the number of internal states that a genetic program needs, as the total number of terminal nodes minus the number of **if-food-ahead** 's. The above conversion algorithm will be applied to evolved S-expression trees in the experiments and we can compare the two types of controllers, FSM and S-expression controllers, in terms of memory states. The FSM built with the algorithm may not be of minimal form in some case, because a certain state may be removed if some sequence of actions are redundant, or if nested and consecutive **if-food-ahead**'s appear in the evolved tree (some motor actions for no food may not be used at all). However, the algorithm will mostly produce a compact form of FSMs.

Table 1(a) is a FSM converted from the genetic program shown in Fig. 2; the genetic programming result has a redundant expression **(progn2) (left) (right)** in the middle of the S-expression and it was not encoded into the FSM. If one looks into the controller in Table 1(a), the behaviour result is almost the same as the Koza's genetic programming result in Fig. 2, even if they are of different formats. Sequential actions were represented as a set of

states in the finite automaton. The result indirectly implies that five internal states are sufficient to collect all 89 pellets of food. We will investigate later what is the minimal amount of memory required to achieve the task. When we evolved FSMs such that ants collect all the food in the grid world, the controllers in Table 1(b)-(c) as well as Table 1(a) were obtained in a small number of generations. The FSM controller in Table 1(a) takes 405 time steps to pick up all 89 pellets of food, while Table 1(b) and Table 1(c) controllers need 379 time steps and 356 time steps, respectively. As an extreme case, a random exploration in the grid would collect all the food if there is a sufficient time available. Thus, the efficiency, that is, the number of time steps needed to collect all the food can be a criterion for better controllers. In this paper, we will consider designing efficient controllers with a given memory structure and explore the relation between performance and the amount of memory.

2.3 Other control structures

Recurrent neural networks have been used to tackle the artificial ant problem (Jefferson et al., 1991, Angeline et al., 1994). The evolved network had a complex interconnectivity and dynamic control process. The attractors and transient states are often hardly identifiable. The amount of memory needed for the problem is not easily quantifiable with the recurrent neural structure, although it can solve the problem through its dynamic process.

A0 = A2 X NoFood x *A_2*
A1 = A2 + 1.434651 / NoFood
A2 = 0.852099 / NoFood
A3 = A2 / (-0.198859 + A2 + 0.200719) x *(-0.696450(A1- A3) / A3)*

Figure 3. An example of multiple interacting programs; *NoFood* is a sensor and *A0, ..., A3* are the actions for no operation, right, left and move, respectively. (Angeline, 1998)

For another memory structure, a concept of multiple interacting programs has been developed to represent state information (Angeline, 1998). A program consists of several symbolic expressions evolved, which can refer to the output of a symbolic equation. A set of symbolic equations corresponds to a set of dynamic state equations, and the program is similar to recurrent neural networks in that the output of one equation can be connected to the inputs of the other equations. Fig. 3.is a program example for the Santa Fe trail problem which has one sensor and four output actions (Angeline, 1998).
The above control structures, recurrent neural networks and multiple interacting programs, have an advantage of representing a complex dynamic operation, but their representations are not quantifiable in terms of internal states. Especially recurrent neural networks have been popularly used in many agent problems with non-Markovian environment (Nolfi and Floreano, 2000), but it would be a difficult task to identify the internal states directly or recognize the relevance and role of internal memory. In contrast, finite state machines and Koza's genetic programming can quantify the amount of internal memory needed for a given agent problem and allow us to analyze the role of internal states on the behaviour performance. In the experiments, we will show this quantitative approach and compare the two different control structures.

3. Evaluation of Fitness Distribution

3.1 *t*-distribution

Student's *t*-test is useful to evaluate the fitness distribution by observing a small number of samples (Cohen, 1995). By calculating the mean and standard deviation over the sample set, the actual mean would be estimated in the range

$$[x - t_{(n,\varepsilon/2)}\ \sigma/sqrt(n),\ x + t_{(n,\varepsilon/2)}\ \sigma/sqrt(n)]$$

where n is the number of samples, x is the mean of samples, $1-\varepsilon$ is a confidence level, and $t_{(n,\varepsilon/2)}$ is the coefficient to determine the interval size, which depends on the number of samples, n. The population standard deviation is unknown, so it is estimated from the sample standard deviation $\sigma/sqrt(n)$, where σ is the standard error of samples.

If there are two collections of performance samples over different strategies, we first draw the confidence ranges for the two sets of samples by the above equation. Then we can statistically determine if one strategy is better than the other. The question reduces to the problem of whether the likely ranges of the population mean overlap. If the ranges overlap each other, we cannot reject the null hypothesis that there is no difference between the two distributions. If the two intervals do not overlap, there is a (1–ε)100% chance that the two means come from different populations and the null hypothesis should be rejected.

The *t*-test is commonly used to determine confidence intervals or critical region for hypothesis tests. Evolutionary approaches are stochastic and the performance results are often demonstrated with the average performance or confidence intervals by the *t*-test. However, the fitness samples in evolutionary computation rarely follow a normal distribution. The above estimation of the population mean may deviate from the actual performance. A set of samples including many outliers tend to inflate the variance and depress the mean difference between a pair of groups as well as the corresponding statistical significance of the *t*-statistic (Miller, 1986). Thus, we suggest another performance measure based on beta distribution.

3.2 Wilcoxon rank-sum test

The Wilcoxon rank-sum test is a non-parametric comparison over two sets of samples and it is based on the rank of the samples (Wild and Seber, 1999). If there are n_a and n_b observations available for the two groups, respectively, the test first ranks the $n_a + n_b$ observations of the combined sample. Each observation is given its own rank in ascending order, that is, starting with 1 for the smallest value. If there is any tie score of observations, the observations are assigned the average rank. The Wilcoxon rank-sum test calculates, r_s , the sum of the ranks for observations which belong to one sample group. When both n_a and n_b are larger than 10, we can assume the distribution of r_s follows a normal distribution. Thus, we can make a significance test over the difference of the two sample distributions by calculating the probablity,

$$\Pr(X > r_s) = \Pr(Z > (w_A - \mu_A)/\sigma_A)), \quad \text{or} \quad \Pr(X < r_s) = \Pr(Z < (w_A - \mu_A)/\sigma_A))$$

where $\mu_A = n_a/(n_a + n_b + 1)/2$, $\sigma_A = (n_a n_b (n_a + n_b + 1)/12)^{1/2}$, X is the distribution of a rank-sum, and Z indicates the normal distribution. If the above probability is less than $\varepsilon/2$, then we can say that a group of samples is larger than the other with confidence level $(1-\varepsilon)$. It rejects the null hypothesis that there is no difference between the two samples.

This Wilcoxon test can be applied to any distribution of samples, regardless of whether it is normal or not, and as well it is not much sensitive to outliers.

3.3 Beta distribution model

The *t*-test provides a possible range of the average performance for a given strategy or control structure. If there exists a decision threshold to evaluate the performance, we can score a trial run as success or failure. In this case we can consider the success rate as a performance measure. Estimation of success rate can be achieved by empirical data, that is, we can take a finite number of trial runs, and count the number of successful runs. Then the relative frequency of success can be an estimate of the success rate. However, this measure does not reflect the total number of runs. The estimated rate may have large deviation from the real success rate when the number of runs is small. For instance, one successful run out of four trial runs should have a different analysis and meaning with 10 successful runs out of 40 runs, although the relative frequency is the same. Thus, we will explore how to estimate the success rate more accurately.

When it is assumed that $\alpha+\beta$ experiments experience α successes and β failures, the distribution of success rate p can be approximated with a beta distribution. The probability distribution of success rate can be obtained with Bayesian estimation (Ross, 2000), which is different from the maximum likelihood estimation of $p = \alpha / (\alpha+\beta)$. The beta probability density function for the success rate is given by

$$f(p, \alpha, \beta) = p^{\alpha}(1-p)^{\beta} / B(\alpha+1, \beta+1)$$

where $B(\alpha+1, \beta+1) = \Gamma(\alpha+1)\,\Gamma(\beta+1) / \Gamma(\alpha+\beta+2)$ and $\Gamma(n+1) = n\Gamma(n)$.

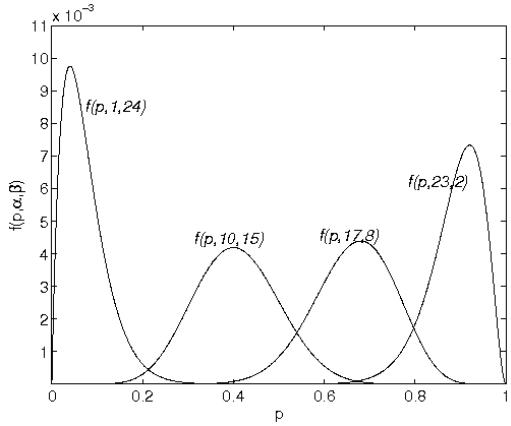


Figure 4. Probabilistic density functions of beta distribution (degree: 25)

Now we want to define confidence intervals about success/failure tests for a given strategy. If the number of experiments is large and the success rate is not an extreme value, then the distribution will be close to a normal distribution. For a small number of samples, the distribution of success probability will follow a beta distribution, not a Gaussian distribution. Fig. 4 shows the probability density function $f(p, \alpha, \beta)$ over a different number of success cases among 25 runs. The beta has very diverse distributions depending on the success cases. For a small or large number of successes, the distribution is far away from

standard normal distribution. Thus, the following numeric approach will be used to estimate confidence intervals. Assume that a random variable X with a beta distribution has the upper bound b_u and the lower bound b_l for confidence limits such that $P(b_l < X < b_u) = 1 - \varepsilon$ and $P(X < b_l) = \varepsilon / 2$. Then we can assert that $[b_l, b_u]$ is a $(1 - \varepsilon)$ 100% confidence interval. If a success probability p is beta-distributed, the confidence limits b_l, b_u can be obtained by solving the following equations:

$$\frac{\varepsilon}{2} = \int_0^{b_u} \frac{p^{\alpha}(1-p)^{\beta}}{B(\alpha+1,\beta+1)}dp, \qquad \frac{\varepsilon}{2} = \int_{b_l}^{1} \frac{p^{\alpha}(1-p)^{\beta}}{B(\alpha+1,\beta+1)}dp \tag{2}$$

The lower and upper bound probability b_l, b_u will decide the 95% confidence interval $[b_l, b_u]$. The above method estimates the success rate with confidence interval, which will be more accurate than the maximum likelihood estimation $p = \alpha / (\alpha+\beta)$. It can also provide better estimation of the computational effort given in Equation (1). By using the interval $[b_l, b_u]$ for $P(m,g)$, the 95% confidence interval of the computational effort will be

$$I(m,z) \in \left[C \cdot \frac{\log(1-z)}{\log(1-b_u)}, \quad C \cdot \frac{\log(1-z)}{\log(1-b_l)} \right]$$

where C is the computing cost for a single run, proportional to mg, and $z=0.95$. If the number of runs to estimate the success rate is small, the confidence interval of the success rate becomes large and so the interval size of the computational effort will increase. The confidence interval of the computational effort provides more precise and meaningful information than a single value estimate that Koza (1992) showed.

Now we compute the computational effort in another respect. We assume that a given experiment is repeatedly run until a successful controller is found. A better strategy or methodology will have a smaller number of runs to obtain a successful controller. Thus, the computational effort (computing time) needed to obtain the first success can be a criterion for performance evaluation. The effort test was suggested by Lee (1998) to compare the performance of different strategies for evolutionary experiments, and he used a measure of the average computing cost needed for the first successful run. In this paper the measure will be extended more rigorously to show the confidence interval of the effort cost.

For a given success rate p over the controller test, the average number of trial runs before the first success run can be calculated as

$$E(X) = \sum_{x=1}^{\infty} xp(1-p)^x = \frac{1-p}{p}$$

where x is the number of trials before the first success. Therefore, the computational effort applied before the first success will be $C(1-p)/p$ where C is a unit computing cost per run (we assume the computational effort only consists of experimental runs who result in failure. If we include the first successful run in the effort, the effort can be estimated with $C_f (1-p)/p + C_s$ where C_f is a failure computing cost and C_s is a success computing cost). If the computing cost per run is variable, we take the averaged cost over multiple runs for an approximate estimation of C. A random variable Y is defined as the amount of trial cost with a success probability p. The expected value of the cost for the random variable Y will be as follows:

$$E(Y) = \int_0^1 \frac{1-p}{p} C \cdot \frac{1}{B(\alpha+1,\beta+1)} p^{\alpha}(1-p)^{\beta} dp$$
$$= \frac{\alpha+\beta+1}{\alpha} C - C = \frac{\beta+1}{\alpha} C$$

If a success rate p has the lower and upper bound probability b_l, b_u by the estimation of confidence interval in Equation (2), the 95% confidence effort cost will be estimated with $[C(1-b_u)/b_u, C(1-b_l)/b_l]$. It is of note that this computational effort is distinguished from Koza (1992)'s estimation given in Equation (1). He assumed that an indefinite number of successful runs, but at least once, are attained with his computational effort, while Lee (1998)'s effort measures the computing cost needed to obtain the first success.

Now we show an example of estimating success rate or computational effort to help understanding of the beta test. When 10 experimental runs are tested for a given task, assume that 9 successes and 1 failure happen for a given strategy. The 95% confidence interval of success rate can be estimated with the two integral equations in Equation (2) (ε= 0.05). If we solve the equations in terms of b_l, b_u, then

$$\frac{0.05}{2} = \int_0^{b_l} \frac{p^9(1-p)^1}{B(9+1,1+1)} dp, \qquad \frac{0.05}{2} = \int_{b_u}^1 \frac{p^9(1-p)^1}{B(9+1,1+1)} dp$$

which gives an approximate solution b_l = 0.587, b_u = 0.977. Thus, [0.587, 0.977] is a confidence interval for a success rate p in this example. The effort cost interval (Lee's effort cost) will be $[C(1-0.977)/0.977, C(1-0.587)/0.587]$ = [.02C, .70C] and its mean $C(\beta+1)/\alpha$ is 0.22C. It is expected in the future experiment that 0.02-0.70 times the computing time of a single run will be spent before the first success with 95% confidence. Koza's computational effort will have a confidence range,

$$I(m,g) \in [C \cdot \frac{\log(1-0.95)}{\log(1-0.977)}, \; C \cdot \frac{\log(1-0.95)}{\log(1-0.587)}] = [0.79C, 3.38C],$$

unlike his estimate of $C \log(1\text{-}0.95) / \log(1\text{-}0.90)$=1.30C with the maximum likelihood probability p=0.90. If there are 90 successful runs out of 100 trials, the success rate will be [0.825, 0.944] and then $I(m,g)$ is in the range of [1.04C, 1.72C]. It shows that we can produce more precise estimation of the effort with more trials.

So far we have shown how to estimate success rate or computational effort related to success rate. By observing a set of fitness samples, we can calculate the confidence interval of success rate, and more experimental runs make the confidence interval of success rate smaller, which means more reliable information of the success rate. To compare strategies, or determine which strategy is more efficient, the confidence interval of success rate or effort cost can be measured for each strategy. Then we can determine statistical significance of the performance difference over strategies. We have shown two kinds of computational efforts which have different meaning of the effort. Both of them depend on the success rate estimation and so they will produce the same decision over the hypothesis test of the performance difference between a pair of groups. In our experiments, the effort cost calculation will use the computing cost needed to obtain the first success, because the computing effort for the first success has a lower level of computing cost and the confidence range is more tight.

4. Experiments

Our evolutionary algorithm will focus on quantifying the amount of memory needed to solve the artificial ant problem. In the ant problem, the fitness function F is defined as follows:

$$F = t_A - \alpha Q_{food}$$

where t_A is the number of time steps required to find all the food, or the maximum allowed amount of time if the ant cannot eat all of them. In the experiments 400 time steps are assigned for each ant agent's exploration, and Q_{food} is the amount of food the ant has eaten. α is a scaling coefficient, which is set to 2 (an ant agent needs 2 time steps on the average to take a food pellet in the nearest neighbours and the coefficient was set to 2). This fitness function considers finding the minimum amount of time to traverse all the food cells. Thus, smaller fitness means better controller. We will use the Santa Fe trail for the target environment. An ant agent may need varying computing time for its fitness evaluation, because t_A varies depending on the time to collect all the food. When an ant succeeds in collecting all the food before 400 time steps, the exploration process can instantly stop not to wait for the whole 400 time steps to complete. This will influence the computing cost and so the computing cost C for a single run will be measured by the averaged CPU run-time over multiple runs.

In this paper, we will test two types of evolving control structures, FSMs and S-expression trees, and compare the performances of ant agents for a given level of memory amount. We will evolve each control structure with a varying number of internal states and analyze fitness samples collected from the evolutionary algorithms.

For the first set of experiments with FSM controllers, the chromosome of a FSM controller is represented as an integer string as described in section 2. One crossover point is allowed only among integer loci, and the crossover rate is given to each individual chromosome, while the mutation rate is applied to every single locus in the chromosome. The mutation will change one integer value into a new random integer value. We used a tournament-based selection method of group size four. A population is subdivided into a set of groups and members in each group are randomly chosen among the population. In each group of four members, the two best chromosomes are first selected in a group and then they breed themselves (more than two chromosomes may have tie rank scores and in this case chromosomes will be randomly selected among the individuals). A crossover over a copy of two best chromosomes, followed by a mutation operator, will produce two new offspring. These new offspring replace the two worst chromosomes in a group. In the experiments, the crossover rate is set to 0.6 and the mutation rate, 0.1, 0.05, or 2 over chromosome length, is applied (the above tournament selection takes a half of the population for elitism, and so the high mutation rate will give more chance of diversity to a new population).

For another memory-based controller, the chromosome of a genetic program is defined as an S-expression tree. The crossover operator on the program is defined as swapping subtrees of two parents. There are four mutation operators available for one chromosome. The first operator deletes a subtree and creates a new random subtree. The subtree to be replaced will be randomly selected in the tree. The second operator randomly chooses a node in the tree and then changes the function (`if-food-ahead` or `progn2`) or the motor action. This keeps the parent tree and modifies only one node. The third operator selects a branch of a subtree and reduces it into a leaf node with a random motor action. It will have the effect of removing redundant subtrees. The fourth operator chooses a leaf node and then

splits it into two nodes. This will assist incremental evolution by adding a function with two action nodes. In the initialization of a population or the recombination of trees, there is a limit for the tree size. The maximally allowable depth of trees is 6 in the initialization, but there is no restriction of the depth after the recombination. The minimum number of leaf nodes is 1 and the maximum number of leaf nodes will be set up as a control parameter in the experiment. If the number of leaf nodes in a new tree exceeds the limit, the tree is mutated until the limit condition is satisfied. The above tournament-based selection of group size four will also be applied to the genetic programming approach. The crossover rate is set to 0.6 and the mutation rate is 0.2.

To quantify the amount of memory, a varying number of states, ranging from 1 state to 20 states, are applied to the artificial ant problem. For each different size of state machines, 50 independent runs with a population size of 100 and 10,000 generations are repeated and their fitness distribution is used for the analysis of memory states. Similar evolutionary experiments are applied to the S-expression controllers with a population size of 500 and 2,000 generations (this parameter setting showed good performance within the limit of 5 $x10^5$ evaluations. This may be due to the fact that genetic programming tends to develop new good offspring through crossover of individuals in a large sized population rather than with the mutation operator (Nolfi and Floreano, 2000)). To compare S-expression and FSM controllers, evolved S-expression controllers are converted into FSMs using the algorithm described in section 2.

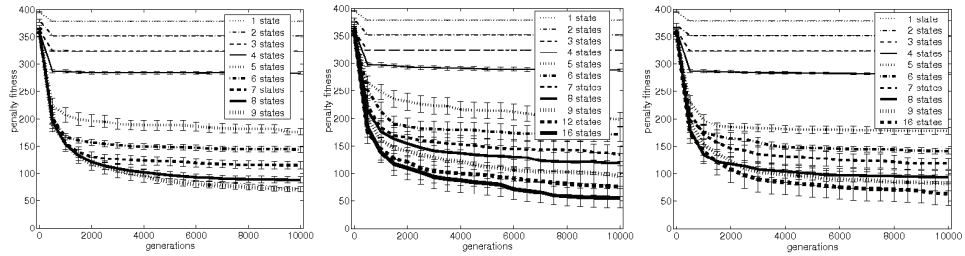
4.1 *t*-test for memory analysis

One of the main interests is how many memory states are required for ants to traverse all the food cells. This should also be addressed together with the question if a given time limit of exploration may influence the number of memory states, that is, how the exploration efficiency is related to the amount of memory. Identifying the characteristics of memory structure will be based on empirical data, that is, fitness samples.

To find out an appropriate number of states needed to reach a given performance level, experiments with a fixed number of states is repeated 50 times. We first applied a mutation rate of 0.1 to the evolution of FSMs. Figure 5(a) shows the average fitness result for each number of memory states. The error bars of 95% confidence intervals are displayed with the average performance over 50 runs by assuming a *t*-distribution. The experiment clearly distinguishes the performances of memory states ranging from one to nine states, although more than nine states were not significantly different from nine states. The error bars for a few number of states are very small, because they have consistently poor performances with too few memory units. Another experiment with a mutation rate of 0.05 produced a slightly different result. The performance comparison by *t* statistic shows that memory states from one to seven had a significant difference of performance. However, there was no significant distinction between seven and eight states or between eight and nine states. We note that for each level of memory amount, the average performance with a mutation rate of 0.1 was better than that with a mutation rate of 0.05, but the best performance or the median with a mutation rate of 0.1 was worse for more than eight states. The mutation rate 0.1 tends to advance a more random search in its evolutionary computation, which was not helpful to find small penalty fitness. In contrast, the mutation rate 0.05 produced better control solutions, although the fitness samples had large variance. Thus, the comparison of the average performance by the *t*-test may lose some information behind the fitness distribution.

We also tested a mutation rate of 2 over chromosome length to give similar chances of mutations to each chromosome. The average performance or best performance for each number of states is mostly better than that with a mutation rate of 0.05 or 0.1. Memory states from one to eight states are distinguishable in performance by *t*-statistic. Still a large number of states have no significant difference in their behaviour performance.

Each number of memory states has its own fitness curve with 95% confidence intervals and we can check whether the fitness samples can reach a given threshold of desirable performance. For example, fitness 222 (= 400 - 89 x 2) or below should be obtained to collect all 89 food pellets. The curves in Figure 5 suggest that four memory states or less are inadequate for the artificial ant problem. Also memoryless reactive controllers fail to cover all the food. We can see in the figure that at least five states are needed to cover all 89 food pellets. Koza's genetic programming result shown in Fig. 2, which used five memory states if a redundant expression is removed, is consistent with the analysis of memory requirements by FSM experiments.



(a) (b) (c)

Figure 5. Fitness test experiments in the Santa Fe trail problem (a) a mutation rate of 0.1 (b) a mutation rate of 0.05 (c) a mutation rate of 2 over chromosome length

We still have a question of how many memory states are required if a time limit is given for ants' successful traversals, or how efficiently ants can collect all 89 food pellets. One can guess that the shorter time limit will lead to more memory states to record environmental states. In Figure 5(a), a low penalty fitness 100 (278 time steps for time limit) can be rarely reached with less than seven states, while more than four states succeeded to cover all the food on the grid. The number of encoded states in FSMs differentiates the fitness performance. Better fitness implies that ants explore the trail environment more efficiently and they need less time to collect all the food. Thus, more states have an ability to explore the environment within a shorter time limit. They memorize the environmental features and trigger the next actions promptly by sequentializing a series of actions. The best evolved controller was a 173 time step controller with fitness –5 (= 173- 89 x 2) as shown in Figure 6. This is comparable to the optimal performance of 165 time steps, in which case the controller memorizes the entire trail completely.

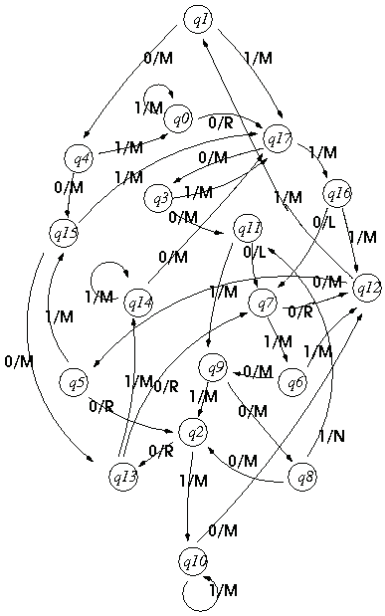
A strategy for an ant agent to collect all the food with five internal states was to look around the environment and move forward if there is no food in the surrounding neighbours. If food is found in one of neighbour cells, the ant immediately moves forward in that direction. The plan needs internal states to take a sequence of actions *(left, right, right, left, move)* or another sequence *(right, left, left, right, move)* with a separate state for one action. If more internal states are given, ants start to utilize environmental features to reduce exploration time, for instance, ants can take three consecutive move actions without looking

around the neighbours after a right or left turn finds food. With a large number of states, more elaborated turns and moves are found in the behaviour of ant agents.

The memory requirement for the task may be determined by the average fitness of multiple runs and its confidence interval, that is, the *t*-statistic (Kim and Hallam, 2001). However, our experiments with a mutation rate of 0.05 or 2 over chromosome length show that the fitness performance has a relatively large variance (error bar) and so the mean difference between a pair of groups is diminished, which influences the statistical significance. Langdon and Poli (1998) showed that the artificial ant problem is highly deceptive, since its fitness landscape is rugged with plateaus, valleys and many local optima. Thus, we suggest a measure of success rate to evaluate fitness samples, based on the beta distribution. This statistic test can be applied to any type of fitness samples regardless of whether they are normally or skewedly distributed.

state	Input 0	Input 1
q0	*q17,R*	*q0,M*
q1	*q4,M*	*q17,M*
q2	*q13,R*	*q10,M*
q3	*q11,M*	*q17,M*
q4	*q15,M*	*q0,M*
q5	*q2,R*	*q15,M*
q6	*q9,M*	*q12,M*
q7	*q12,R*	*q6,M*
q8	*q2,M*	*q11,N*
q9	*q8,M*	*q2,M*
q10	*q12,M*	*q10,M*
q11	*q7,L*	*q9,M*
q12	*q5,M*	*q1,M*
q13	*q7,R*	*q14,M*
q14	*q17,M*	*q14,M*
q15	*q13,M*	*q17,M*
q16	*q7,L*	*q12,M*
q17	*q3,M*	*q16,M*

a)



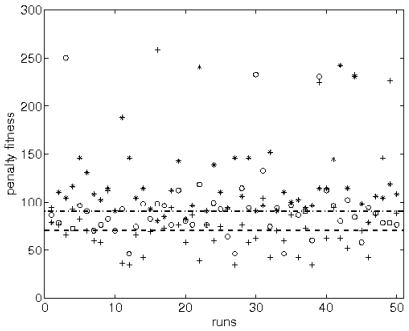
b)

Figure 6. One of the best evolved control structure FSM in the Santa Fe trail problem (a) transition table (b) state diagram (input 1: food-ahead, input 0: no food ahead, output set is L (turn left), R(turn right), M (move forward), N (no-operation))

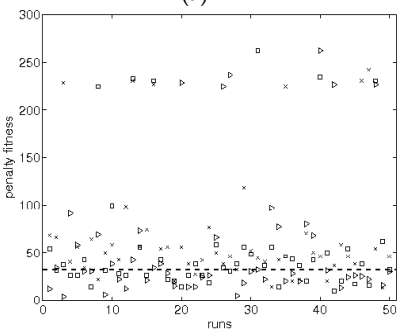
4.2 Wilcoxon rank-sum test for memory analysis

For experiments with a mutation rate of 2 over chromosome length, internal states from one to eight generate distinctive difference of performance and they can be discriminated by the *t*-test as shown in Fig. 5, but internal states from eight to ten or more than ten states are not significantly different. Their confidence interval sizes are relatively large and the intervals

overlap, since the fitness distribution has a large variance as shown in Fig. 7; it should be noted that the fitness samples do not follow a Gaussian distribution and have several outliers. By the deceptive property of fitness, it will be difficult to have normal distribution even with a large number of samples. Thus, the Wilcoxon rank-sum test is applied to the fitness samples for any pair of state machines. The best penalty fitness of four-state machines among 25 trials was 280, and the best fitness of five-state machines ranges from 154 to 274. The four-state and five-state machines were serially positioned in ranking order of fitness samples. The Wilcoxon rank-sum test reveals significance in difference of the two samples. In a similar way, we can estimate the partial order of FSMs with a variety number of states. Figure 8 shows the results with a probability p for each pair of machines. The probability $1-p$ represents the confidence level for the partial order relation. We assume 95% confidence level for the relation and if $1-p$ is less than 0.05, we reject the null hypothesis that there is no performance difference between a pair of two sample groups.



(a)

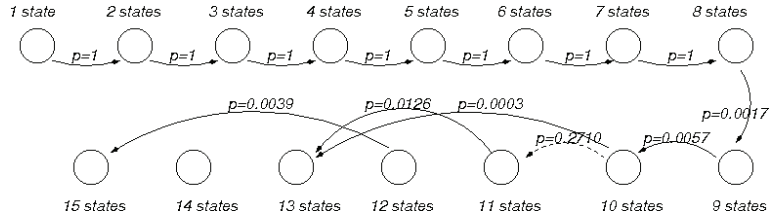


(b)

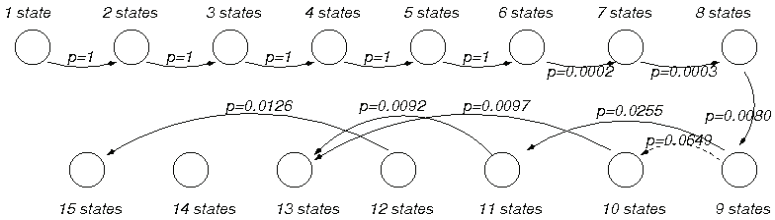
Figure 7. Fitness distribution of FSM controllers (a) with 7 states, 8 states and 9 states (*: 7 states, circle: 8 states, +: 9 states, dotdashed: threshold fitness 90 for success, dashed: threshold fitness 70 for success) (b) with 11 states, 13 states and 15 states (x: 11 states, square: 13 states, triangle: 15 states, dotdashed: fitness 38)

FSMs ranging from one state to 10 states show significant difference among their behaviour performances. The memory effect on the behaviour performance is rather reduced for more than 10 states. For example, a significance level for the difference between 10 states and 11 states or between 11 states and 12 states is not sufficient to reject the null hypothesis. Generally the Wilcoxon rank-sum test show more precision of partial ordering that t-statistic

loses, since it is not sensitive to many outliers. We compared the results with 50 trials and 25 trials for each number of state machines. The 50 trials and 25 trials produced almost the same partial order relations except for the FSMs ranging from 9 to 11 states. The statistic probability p varies depending on the number of trials.



(a)



(b)

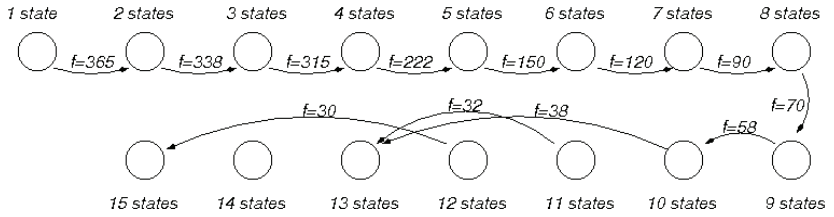
Figure 8. A partial order relation of FSMs with varying number of memory states in terms of behaviour performance by Wilcoxon rank-sum test (a) 50 trials (b) 25 trials (a mutation rate of 2 over chromosome length, solid: significant difference in the Wilcoxon test, dashed: insignificant difference)

4.2 Beta distribution for memory

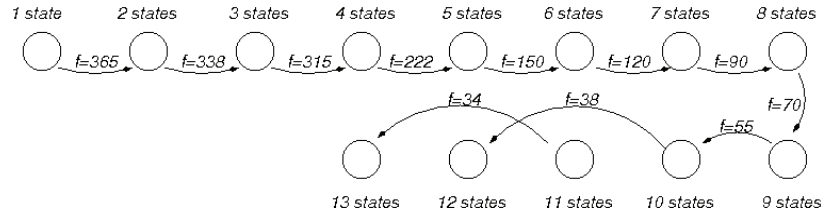
Now we apply the beta distribution analysis to the fitness samples for finite state machines with a variety of internal states. We assume that if there exists any decision threshold of fitness for success/failure for a given pair of controllers such that it causes significant difference of performance, then the control structures are distinguishable in performance. Between seven and eight states, we place a decision threshold of fitness 90 to determine their success/failure. Then we obtain 5 successes among 50 trials for seven internal states and 27 successes for eight states -- see Figure 7. If we calculate the confidence interval of success rate in beta distribution, seven states and eight states have success rates, [0.044, 0.214] and [0.403 0.671], respectively, and their effort costs will be $[C_7(1\text{-}0.214)/0.214,\ C_7\ (1\text{-}0.044)/0.044] = [3.67\ C_7,\ 21.73\ C_7]$ and $[C_8(1\text{-}0.671)/0.671,\ C_8(1\text{-}0.403)/0.403] = [0.49C_8,\ 1.48C_8]$, where C_7, C_8 is the average computing cost for a single run in failure mode, that is, an experimental run which does not reach the fitness 90. Here, it is assumed that the cost C_7, C_8 is almost identical for the two types of state machines. Then the confidence intervals are significantly different and thus we can clearly see the performance difference of the two control structures. Similarly with a decision threshold of fitness 70, we have 6 successes and 22 successes among 50 trials for eight states and nine states, respectively. Their confidence intervals of the success rate are [0.057, 0.239] and [0.311, 0.578], and their effort costs are [$3.19C$, $16.54C$], [$0.73C$, $2.21C$] by assuming that the two types of state machines have the same computing cost. Then the difference of the success rate or effort cost between the two control structures is also significant.

In fact, the computing cost for each run is variable by different exploration time. In the evolutionary experiments the averaged CPU run-time over multiple runs resulting in failure was $C_{f7} = 39.69$ sec, $C_{f8} = 41.96$ sec for seven and eight states, respectively, while the averaged CPU run-time for success was $C_{s7} = 38.20$ sec, $C_{s8} = 38.36$ sec. The difference of the computing costs does not influence the above significance analysis, because the ratio of the costs is close to 1. In the experiment, the average computing cost for a different size of state machines had a similar level. The decision of significance test by the effort cost agreed with that by the success rate.

By the analysis on the experiments with a mutation rate of 2 over chromosome length, we found that finite states ranging from one to ten have distinctive difference in performance. There was no significant difference between 10 states and 11 states, or between 11 states and 12 states. However, FSMs with 13 states showed significantly better performance than FSMs with 10 states. Fig. 9(a) shows a partial order relation of FSMs with varying number of states, which was estimated by the confidence interval of success rate or effort cost. Evolving more than 15 states produced slightly better controllers than evolving 15 states, but the difference among the fitness samples was not significant. In the same procedure of memory analysis, we tested the statistical significance with a smaller number of trials, 25 trials. We could still obtain similar partial order relation among a varying number of states, although the performance difference between 12 states and 15 states became insignificant. It implies that the beta distribution model can be used for performance comparison even with a small number of trials. The above results show that the beta distribution analysis can find significance of performance differences where fitness samples have a large variance due to outliers or where *t*-statistic does not produce useful information. It is noteworthy that the partial order relation result by the beta distribution is exactly the same as that by Wilcoxon test with 50 trials. The result indirectly supports the validity and usefulness of the beta distribution.



(a) 50 trials



(b) 25 trials

Figure 9. A partial order relation of FSMs with varying number of memory states in terms of behaviour performance by beta distribution (a) 50 trials (b) 25 trials (a mutation rate of 2 over chromosome length is applied and the arrow label indicates the threshold fitness for success/failure)

4.3 Genetic programming approach

For another type of memory-based controllers, we used S-expression controllers. Let n_t, n_f, n_s the number of terminal nodes, the number of the function **if-food-ahead**'s, and the number of internal states in an S-expression tree, respectively. By the conversion algorithm described in section 2, we can build an FSM such that the number of states is $n_s = n_t - n_f$. Fig. 10 shows a conversion example for an evolved S-expression tree. The states {*q0, q1, q5, q6*} in Fig. 10(b) have different state transitions or motor actions on the input condition and they correspond to the operation of the function **if-food-ahead**. The other states define the same transition and action on the input 0 and 1, which is a copy operation of **progn2**. The conversion algorithm produces a compact form of FSM and helps tracing internal states that the genetic program uses. In this paper we use two control parameters, n_t, n_s to evolve S-expression trees. Once a control parameter is set up for the evolutionary experiments, for instance, the number of terminal nodes is defined, the size of evolved trees should be within the limit size. If an evolved tree is over the limit, the tree should be re-generated by mutation to satisfy the condition.

```
(if-food-ahead
  (move)
  (progn2 (right)
          (progn2
            (if-food-ahead
                  (progn2
                        (progn2 (move) (move))
                        (move))
                (progn2
                        (progn2 (left) (left))
                        (if-food-ahead
                          (move) (right))))
          (if-food-ahead
                (progn2 (move) (move))
                (move)))))
```

a)

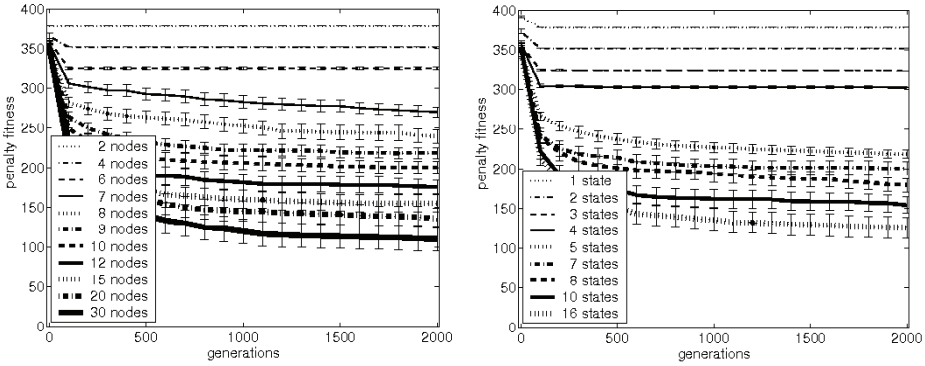
state	Input 0	Input 1
q0	*q1,R*	*q0,M*
q1	*q4,L*	*q2,M*
q2	*q3,M*	*q3,M*
q3	*q6,M*	*q6,M*
q4	*q5,L*	*q5,L*
q5	*q6,R*	*q6,M*
q6	*q0,M*	*q7,M*
q7	*q0,M*	*q0,M*

b)

Figure 10. An example of genetic programming result (a) evolved S-expression (n_t=*12*, n_f=*4*) (b) converted FSM

In the first experiment, a varying number of terminal nodes were tested by using the number of terminal nodes as a control parameter. A large number of terminal nodes can expect more sequence of actions and thus produce better performance. Figure 11(a) shows the average fitness performance with its 95% confidence range by *t*-statistic for a given number of terminal nodes. Evolving 20 terminal nodes and 30 terminal nodes had 3, 22 cases to reach the fitness 90 or below, respectively, which are similar to the performance of 7 states (5 successes) and 8 states (27 successes) among the FSM controllers in Fig. 5. However, it is not easy to compare directly the performances of FSM and S-expression controllers, since they should have the same criterion for comparison. The best evolved tree for each run often follows the rule that the number of terminal nodes can be approximated by 1.5 times the number of states (see Fig. 12), when the tree is converted into the corresponding FSM structure and the internal states are counted. The best trees with 20 terminal nodes had a range of 9-15 states, and the performance was significantly lower than the performances of

FSM controllers evolved with the same range of states. It indirectly entails that evolving FSMs can provide a better solution to encode internal memory.



(a) (b)

Figure 11. Genetic programming result (a) evolve controllers with a fixed number of terminal nodes (b) evolve controllers with a fixed number of states

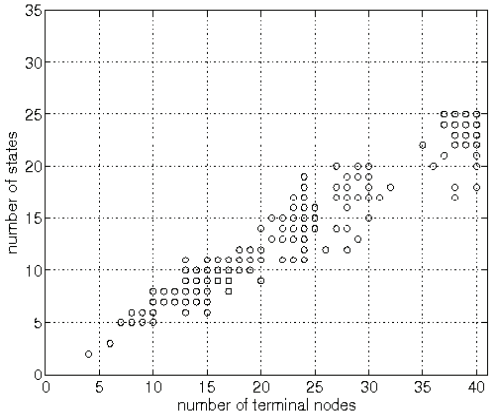
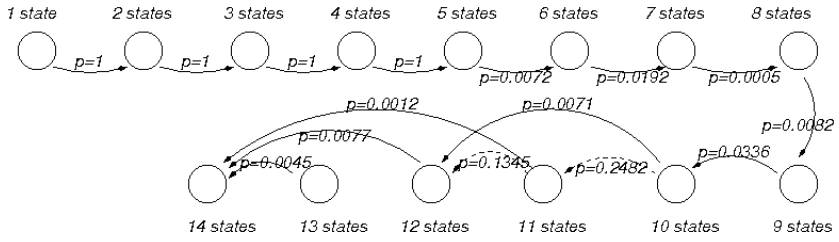


Figure 12. Relation between terminal nodes and internal states (genetic programming controllers are evolved with a varying number of terminal nodes)

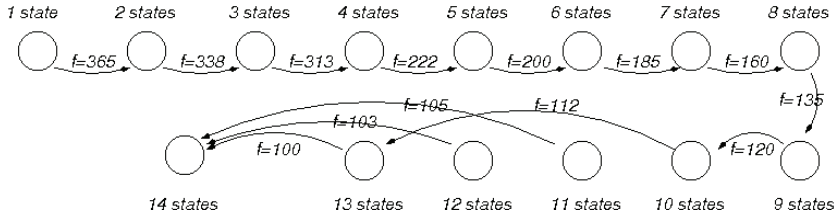
For the next experiment, we set up the number of states as a control parameter. In a similar process as above, we test a varying number of states. If an evolved tree includes more states than the limit, the tree will be mutated until the limit condition is satisfied. In addition, we set the maximum number of terminal nodes for evolving trees to $1.7\ n_s$ for each number of states, n_s , using the above relation rule between terminal nodes and states (Fig. 12). Without the restriction on terminal nodes, the evolving trees encountered bloat, degrading the performance. As shown in Fig. 11(b), the internal states play a critical role on the performance improvement, and the fitness performance is enhanced with more internal states. However, for a given number of states, the genetic programming result is mostly worse than the FSM evolution result shown in Fig. 5. The best fitness that the genetic programming achieved for the overall experiments was 44, and it had a similar performance

with the FSM evolution only for a few sets of internal states. Evolving directly the FSM structure tends to produce more efficient controllers and its performance level is significantly better. The S-expression is a procedural program arranging a sequence of actions, while FSMs can not only encode a sequence of actions, but also they have more dynamic features in representation by allowing flexible transitions and actions from state to state.

By the fitness distribution of genetic programming controllers, we built a partial order relation among memory structure as shown in Figure 13. The Wilcoxon rank-sum test and beta distribution test produce almost the same diagram of partial order relations except for the case between 10 states and 12 states. One reason for this is that the Wilcoxon test measures the rank sum over all the observations which belong to one sample group, whilst the beta distribution focuses on what level of good fitness performance one sample group reach. When the diagram is compared with the partial order graph by the FSM result (Fig. 9), the threshold level for the relation is changed, but roughly keeps the relation structure. It seems that a large number of states have more variation on the relation result. As a matter of fact, the lattice diagram for memory analysis is extracted from empirical data which depends on the corresponding control structure. It only shows the partial order information among memory structure with a given confidence level, but it does not mean that the corresponding structure guarantees a given level of performance or it cannot achieve better performance than the threshold level. More trial runs will support more reliable information of the performance level or the partial order relation.



(a)



(b)

Figure 13. A partial order relation of S-expression trees with varying number of states in terms of behaviour performance (a) Wilcoxon rank-sum test (solid: significant difference, dashed: insignificant difference) (b) beta distribution test (the arrow label indicates the threshold fitness for success/failure)

So far we showed the distribution of success rate or computational effort to measure the performance difference between a pair of sample groups. The significance test result of the success rate is equivalent to that of the effort cost if almost the same computing cost is spent for each evolutionary run. At this point the analysis of the effort cost is not more helpful than the analysis of success rate. However, the information of computational effort can be used to understand the evolutionary process for a given problem. Table 2 shows the number of successful runs among 50 trials and the computational effort for each configuration.

To reach the fitness level 150, FSM controllers obtained 17 successes with 6 internal states and 10^5 evaluations (2,000 generations). More generations, for example, 5,000 generations and 10,000 generations produced more successes as expected. The unit cost C_2, C_5 , C_{10} is the computing cost for 10^5, 2.5 x 10^5, 5 x 10^5 evaluations and the costs can be approximated with $C_{10} = 2C_5 = 5C_2$. Then the confidence intervals for 10^5, 2.5 x 10^5, 5 x 10^5 (2,000, 5,000 and 10,000 generations) become $[1.09C_2, 3.46C_2]$, $[1.69C_2, 5.09C_2]$, $[1.74C_2, 5.40C_2]$, respectively. Then the experiments with 2,000 generations have the confidence interval with the smallest effort level, whose range is also narrow. Thus, we can recommend 2,000 generations for the future experiments. For 11 states with a desirable fitness of 50, 10^5, 2.5 x 10^5, 5 x 10^5 evaluations produce the confidence intervals $[5.16C_2, 44.91C_2]$, $[2.51C_2, 7.86C_2]$ and $[3.12C_2, 9.39C_2]$, respectively. For this case, 2.5 x 10^5 evaluations (5,000 generations) would be a better choice for evolution.

Parameters			genome evaluations					
pop.	states	fitness	$g \le 10^5$		$g \le 2.5 \times 10^5$		$g \le 5 \times 10^5$	
100	5	222	48	$[0.01C_2, 0.16C_2]$	48	$[0.01C_5, 0.16C_5]$	48	$[0.01C_{10}, 0.16C_{10}]$
100	6	150	17	$[1.09C_2, 3.46C_2]$	23	$[0.67C_5, 2.03C_5]$	31	$[0.35C_{10}, 1.08C_{10}]$
100	7	120	22	$[0.73C_2, 2.21C_2]$	33	$[0.29C_5, 0.92C_5]$	37	$[0.19C_{10}, 0.65C_{10}]$
100	8	150	45	$[0.05C_2, 0.27C_2]$	46	$[0.03C_5, 0.23C_5]$	47	$[0.02C_{10}, 0.19C_{10}]$
100	8	100	21	$[0.79C_2, 2.41C_2]$	32	$[0.32C_5, 1.00C_5]$	41	$[0.11C_{10}, 0.45C_{10}]$
100	9	70	7	$[2.81C_2, 13.24C_2]$	17	$[1.09C_5, 3.46C_5]$	22	$[0.73C_{10}, 2.21C_{10}]$
100	10	50	2	$[6.43C_2, 80.30C_2]$	10	$[2.02C_5, 7.86C_5]$	19	$[0.93C_{10}, 2.87C_{10}]$
100	11	50	3	$[5.16C_2, 44.91C_2]$	18	$[1.00C_5, 3.14C_5]$	24	$[0.62C_{10}, 1.88C_{10}]$
100	15	25	1	$[8.57C_2, 208.21C_2]$	6	$[3.19C_5, 16.54C_5]$	19	$[0.93C_{10}, 2.87C_{10}]$

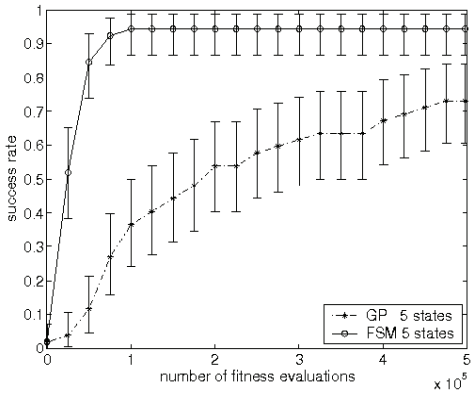
(a)

Parameters			genome evaluations					
pop.	states	fitness	$g \le 10^5$		$g \le 2.5 \times 10^5$		$g \le 5 \times 10^5$	
500	5	222	18	$[1.00K_{0.4}, 3.14K_{0.4}]$	29	$[0.42K_1, 1.26K_1]$	37	$[0.19K_2, 0.66K_2]$
500	6	200	15	$[1.28K_{0.4}, 4.23K_{0.4}]$	23	$[0.68K_1, 2.04K_1]$	28	$[0.45K_2, 1.36K_2]$
500	7	180	5	$[3.67K_{0.4}, 21.51K_{0.4}]$	12	$[1.67K_1, 5.98K_1]$	13	$[1.52K_2, 5.29K_2]$
500	8	150	5	$[3.67K_{0.4}, 21.51K_{0.4}]$	6	$[3.19K_1, 16.54K_1]$	9	$[2.23K_2, 9.18K_2]$
500	9	150	7	$[2.81K_{0.4}, 13.24K_{0.4}]$	12	$[1.67K_1, 5.98K_1]$	17	$[1.09K_2, 3.46K_2]$
500	10	120	5	$[3.67K_{0.4}, 21.51K_{0.4}]$	10	$[2.02K_1, 7.86K_1]$	11	$[1.83K_2, 6.82K_2]$
500	11	120	7	$[2.81K_{0.4}, 13.24K_{0.4}]$	13	$[1.52K_1, 5.29K_1]$	15	$[1.28K_2, 4.23K_2]$

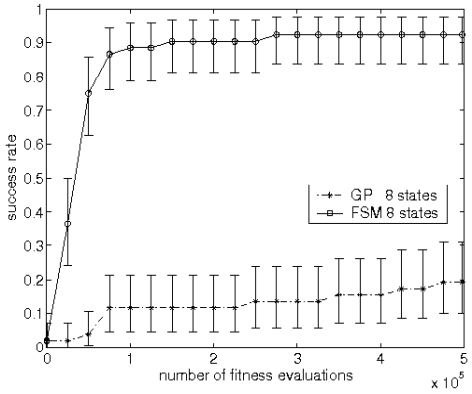
(b)

Table 2. Effort costs for exploration time with number of genome evaluations (a) evolving FSM controllers (b) evolving S-expression trees; C_i, K_i is the basic unit effort cost of a single run which fails to reach the corresponding fitness (C_i, K_i slightly varies on different fitness levels, but for simplicity the same notation is used). Each pair of values represent the number of successes among 50 experiments and its 95% confidence interval for the effort cost

We can compare the computational efforts for different types of controllers or different algorithms. The analysis of the effort cost will be useful especially when the algorithms have different CPU run-time for an evolutionary run. For instance, the FSM controller with eight states and 10^5 evaluations (2,000 generations) has a confidence interval $[0.05C_2, 0.27C_2]$ to obtain the fitness 150, while the genetic programming for eight states needs a computing cost $[3.67K_{0.4}, 21.51\ K_{0.4}]$ to obtain the first success with 95% chance. If we know the computing costs for a single run, C_2 and $K_{0.4}$ we can take the significance test over the two strategies by comparing the confidence intervals. Indeed, the ratio of the average CPU run-time between the FSM and the genetic programming ($K_{0.4}$ / C_2) was 1.23 in the experiments. The comparison result implies that the FSM controllers produce more efficient results than the genetic programming controllers. This is confirmed again in other pairwise tests for small fitness in Table 2.



(a)



(b)

Figure 14. Run-time distribution with genetic programming and FSM controllers (the average and 95% confidence range of success rate in the beta distribution are displayed) (a) evolving five internal states (threshold fitness: 222) (b) evolving eight internal states (threshold fitness: 150)}

Hoos and Stuetzle (1998) studied a run-time distribution to compare different algorithms or determine parameter settings. The distribution shows the empirical success rate depending on varying run time. In our experiments, the number of trial runs is relatively small and so the beta distribution of success rate was applied to the run-time distribution as shown in Fig. 14, where the number of fitness evaluations was used instead of the actual CPU run-time. The FSMs with five and eight states showed significantly better performance in the run-time process. Generally, FSMs show more discriminative performance for a large number of states.

5. Discussion

The number of states in finite state machines in the experiments may not be exactly the same as the number of states that the evolved controllers actually use for exploration. It specifies a maximum limit over the number of finite states. Especially for a large number of states, controllers that do not use all memory states are sometime evolved even though a given maximum memory limit is specified. The same can be true of the genetic programming structure. When the maximum number of terminal nodes was set up for evolutionary runs, some best controllers used a smaller number of nodes than the limit size, or had a redundant expression. Thus, our analysis of memory states may have a little discrepancy with the actual usage of memory.

Genetic programming has a high-level representation feature with a procedural program. When an S-expression is translated into a finite automaton, it has a main loop for repeating the action sequence. It often has a sequential process among internal states until the end of program is reached. In contrast, the Mealy machine notation allows transition loops among internal states. Evolving the FSM controllers can create such loops for in-between states (from state to state) and more conditional transition branches. The flexible representation of the Mealy machine provides more dynamic property for a given number of states. The performance difference between the two types of controllers is due to the characteristics of representation.

To discriminate the performances of a varying number of internal states, the beta distribution of success rate or computational effort was used. We believe that the success rate is a better criterion for this application, because we are more interested in the on-off decision of the quality of controllers with a given evolutionary setting rather than efficient development of controllers. The computational effort can be more effective when strategies to be compared have different computing costs or when the efficiency is a major criterion in the evolutionary experiments. An assumption for the suggested significance test of computational effort is that each single run has almost the same level of computing cost. If each run may have a significantly different computing cost, the estimated computational effort based on success rate would have a deviation from the actual effort. The run-time distribution, that is, the curve of success rate for variable computing cost provides the characteristics of a given algorithm and we can easily observe the transition of performance with run-time. The run-time distribution with its confidence range would be a useful tool to compare different algorithms.

In the evolutionary computation research, the performance comparison among evolutionary algorithms has often used the average performance over fitness samples or *t*-statistic. We argue that the comparison without observing the fitness distribution may not notice significant difference. The beta distribution analysis or Wilcoxon rank-sum test would

provide more precise information than *t*-distribution when the number of samples is small or the fitness samples are influenced by the deceptive property of a given problem. We applied the Wilcoxon rank-sum test and the beta distribution for partial order relations between a varying number of internal states with FSMs. The beta distribution test with 50 trials produced the same significance result as the well-known Wilcoxon test and it indirectly supports the validity of the suggested beta distribution. An advantage of the beta distribution is that it can estimate the computational effort and success rate without difficulty, as well as do significance test over a pair of sample groups. It attends to how many observations for a desirable level of performance are found consistently with a given strategy rather than considers the whole distribution of observations. If an evolutionary search problem is deceptive, a skewed or bimodal distribution of fitness samples can be observed. The above results imply that the non-parametric tests over the distribution of fitness samples would be more effective.

The memory analysis and methods suggested in this paper can be applied to agent behaviours in a grid world. The tested ant problem has a restricted set of sensory configurations and motor actions. However, the agents in the real world, which often includes a variety of sensory apparatus and motor actions, have different characteristics or behaviours. Noisy sensor readings may need more dynamic configurations or structures. The FSM control structure has a representation problem for a large number of input conditions and a large scale of motor actions, although the amount of internal memory is easily quantifiable. Thus, the current approach may have a limitation in real-world problems. In the experiments we tested a single environment, but it may need more variety of environmental configurations to design robust controllers. Then the memory condition for the ant problem will be changed. We leave this work for future study.

6. Conclusion

The artificial ant problem is an agent task to model ant trail following in a grid world, and the environment has a perceptual aliasing problem. The ant problem is a good example to see memory effects in behavior performance. An artificial ant agent must seek all the food lying in a grid and follow the irregular trail for the food. Relatively many memory states are required to remember environmental features. Its performance is measured by how many time steps are required for an ant agent to collect all food on a trail.

We applied two types of memory encoding controllers, finite state machines and genetic programming controllers for the analysis of internal memory, where the genetic programming controllers are transformable into FSMs. We first explored a systematic analysis over the usage of internal memory, based on statistical significance test and tried to identify the role of internal states. The internal states needed for a given threshold performance are easily quantifiable with FSMs. We obtained a partial order of memory states necessary with a variety of performance levels. The results show that the ant problem needs at least five internal states. To develop more efficient strategies (collecting all food in a shorter time) in the artificial ant problem, agents need more memory states according to the memory analysis and also there is a limit level of memory over which the performance is rarely improved. Using the suggested approach, we can identify the role of internal states or observe the relevance of memory to agent behaviours. This will help us understand the characteristics of agent behaviours by decomposing the behaviours in terms of internal states.

Three different statistic tests, *t*-test, Wilcoxon rank-sum, and beta distribution, were applied to discriminate the performance difference among a varying number of internal states. The beta distribution test has a good precision of significance test, and its test result is similar to that of the Wilcoxon test. In many cases, the beta distribution test of success rate was useful where the *t*-test could not discriminate the performance. The beta distribution test based on sampling theory has an advantage on analyzing the fitness distribution with even a small number of evolutionary runs, and it has much potential for application as well as provide the computational effort. In addition, the method can be applied to test the performance difference of an arbitrary pair of methodologies. The estimation of computational effort provides the information of an expected computing time for success, or how many trials are required to obtain a solution. It can also be used to evaluate the efficiency of evolutionary algorithms with different computing time.

We compared genetic programming approach and finite state machines, and the significance test with success rate or computational effort shows that FSMs have more powerful representation to encode internal memory and produce more efficient controllers than the tree structure, while the genetic programming code is easy to understand.

7. References

P.J. Angeline, G.M. Saunders, and J.B. Pollack (1994). An evolutionary algorithm that constructs recurrent neural networks, *IEEE Trans. on Neural Networks*, 5(1): pp. 54-65.

P.J. Angeline (1998). Multiple interacting programs: A representation for evolving complex Behaviors, *Cybernetics and Systems*, 29(8): pp. 779-806.

D. Ashlock (1997). GP-automata for dividing the dollar, *Genetic Programming 97*, pp. 18-26. MIT Press

D. Ashlock (1998). ISAc lists, a different representation for program induction, *Genetic Programming 98*, pp. 3-10. Morgan Kauffman.

B. Bakker and M. de Jong (2000). The epsilon state count. *From Animals to Animats 6: Proceedings of the Sixth Int. Conf. on Simulation of Adaptive Behaviour*, pp. 51-60. MIT Press

K. Balakrishnan and V. Honavar (1996). On sensor evolution in robotics. Genetic Programming 1996: *Proceedings of the First Annual Conference*, pages 455--460, Stanford University, CA, USA. MIT Press.

D. Braziunas and C. Boutilier (2004). Stochastic local search for POMDP controllers. *Proc. of AAAI*, pages 690--696

S. Christensen and F. Oppacher (2002). An analysis of Koza's computational effort statistics, *Proceedings of European Conference on Genetic Programming*, pages 182-191.

P. R. Cohen (1995), *Empirical methods for artificial intelligence*, MIT Press, Cambridge, Mass., 1995.

M. Colombetti and M. Dorigo (1994), Training agents to perform sequential behavior, *Adaptive Behavior*, 2 (3): 305-312.

J. Elman (1990). Finding structure in time. *Cognitive Science*, 14: 179-211.

L.J. Fogel, A.J. Owens, and M.J. Walsh (1996), *Artificial intelligence through simulated evolution*, Wiley, New York, 1966.

H.H. Hoos and T. Stuetzle (1998). Evaluating LasVegas algorithms - pitfalls and remedies, *Proceedings of the 14th Conf. on Uncertainty in Artificial Intelligence*, pages 238--245. Morgan Kaufmann

H.H. Hoos and T. Stuetzle (1999). Characterising the behaviour of stochastic local search, *Artificial Intelligence*, 112 (1-2): 213--232.

J.E. Hopcroft and J.D. Ullman (1979). *Introduction to automata theory, languages, and computation*. Addison Wesley, Reading, MA.

D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang (1991). Evolution as a theme in artificial life, *Artificial Life II*. Addison Wesley.

D. Kim and J. Hallam (2001). Mobile robot control based on Boolean logic with internal Memory, *Advances in Artificial Life, Lecture Notes in Computer Scien*ce vol. 2159, pp. 529-538.

D. Kim and J. Hallam (2002). An evolutionary approach to quantify internal states needed for the Woods problem, From Animals to Animats 7, Proceedings of the Int. Conf. on the Simulation of Adaptive Behavior}, pages 312-322. MIT Press

D. Kim (2004). Analyzing sensor states and internal states in the tartarus problem with tree state machines, *Parellel Problem Solving From Nature 8, Lecture Notes on Computer Science* vol. 3242, pages 551-560.

D. Kim (2006). Memory analysis and significance test for agent behaviours, *Proc. of Genetic and Evolutionary Computation Conf. (GECCO),* pp. 151-158.

Z. Kohavi (1970). *Switching and Finite Automata Theory*, McGraw-Hill, New York, London.

J. R. Koza (1992). Genetic Programming, MIT Press, Cambridge, MA.

W.B. Langdon and R. Poli (1998). Why ants are hard, *Proceedings of Genetic Programming*.

P.L. Lanzi.(1998).An analysis of the memory mechanism of XCSM, Genetic Programming 98, pages 643--651. Morgan Kauffman

P.L. Lanzi (2000). Adaptive agents with reinforcement learning and internal memory, *From Animals to Animats 6: Proceedings of the Sixth Int. Conf. on Simulation of Adaptive Behaviour*, pages 333-342. MIT Press.

W.-P. Lee (1998) *Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots*, Ph. D. dissertation, University of Edinburgh.

L. Lin and T. M. Mitchell (1992). Reinforcement learning with hidden states, *From Animals to Animats 2: Proceedings of the Second Int. Conf. on Simulation of Adaptive Behaviour*, pages 271--280. MIT Press.

A.K. McCallum (1996). *Reinforcemnet Learning with Selective Perception and Hidden State*, Ph.D. dissertation, University of Rochester.

N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling (1999). Learning finite-state controllers for partially observable environments. *Proc. of the Conf. on UAI*, pages 427—436.

J.H. Miller.The coevolution of automata in the repeated prisoner's dilemma, *Journal of Economics Behavior and Organization*, 29(1): 87-112.

Jr. R. Miller (1986). *Beyond ANOVA, Basics of Applied Statistics*, John Wiley & Sons, New York.

J. Niehaus and W. Banzhaf (2003). More on computational effort statistics for genetic programming. *Proceedings of European Conference on Genetic Programming*, pages 164-172.

S. Nolfi and D. Floreano (2000). Evolutionary Robotics : The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press, Cambridge, MA.

L. Peshkin, N. Meuleau, L.P. Kaelbling (1999). Learning policies with external memory, *Proc. of Int. Conf. on Machine Learning*, pp. 307-314, 1999

S.M. Ross (2000*). Introduction to Probability and Statistics for Engineers and Scientists*. Academic Press, San Diego, CA, 2nd edition.

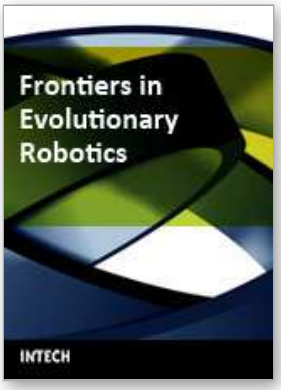
A. Silva, A. Neves, and E. Costa (1999). Genetically programming networks to evolve memory mechanism, *Proceedings of Genetic and Evolutionary Computation Conference.*

E.A. Stanley, D. Ashlock, and M.D. Smucker (1995). Iterated prisoner's dilemma game with choice and refusal of partners, *Advances in Artificial Life : Proceedings of European Conference on Artificial Life*.

A. Teller (1994). The evolution of mental models, Advances in Genetic Programming. MIT Press.

C. Wild and G. Seber (1999). *Chance Encounters: A First Course in Data Analysis and Inference*. John Wiley & Sons, New York.

S.W. Wilson (1994). ZCS: A zeroth level classifier system, *Evolutionary Computation*, 2 (1): 1-18.



Frontiers in Evolutionary Robotics

Edited by Hitoshi Iba

ISBN 978-3-902613-19-6
Hard cover, 596 pages
Publisher I-Tech Education and Publishing
Published online 01, April, 2008
Published in print edition April, 2008

This book presented techniques and experimental results which have been pursued for the purpose of evolutionary robotics. Evolutionary robotics is a new method for the automatic creation of autonomous robots. When executing tasks by autonomous robots, we can make the robot learn what to do so as to complete the task from interactions with its environment, but not manually pre-program for all situations. Many researchers have been studying the techniques for evolutionary robotics by using Evolutionary Computation (EC), such as Genetic Algorithms (GA) or Genetic Programming (GP). Their goal is to clarify the applicability of the evolutionary approach to the real-robot learning, especially, in view of the adaptive robot behavior as well as the robustness to noisy and dynamic environments. For this purpose, authors in this book explain a variety of real robots in different fields. For instance, in a multi-robot system, several robots simultaneously work to achieve a common goal via interaction; their behaviors can only emerge as a result of evolution and interaction. How to learn such behaviors is a central issue of Distributed Artificial Intelligence (DAI), which has recently attracted much attention. This book addresses the issue in the context of a multi-robot system, in which multiple robots are evolved using EC to solve a cooperative task. Since directly using EC to generate a program of complex behaviors is often very difficult, a number of extensions to basic EC are proposed in this book so as to solve these control problems of the robot.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

DaeEun Kim (2008). A Quantitative Analysis of Memory Usage for Agent Tasks, Frontiers in Evolutionary Robotics, Hitoshi Iba (Ed.), ISBN: 978-3-902613-19-6, InTech, Available from: http://www.intechopen.com/books/frontiers_in_evolutionary_robotics/a_quantitative_analysis_of_memory_usage_for_agent_tasks

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821