Studying complexity of simple agents through the processes of mutation and natural selection

Andrew Martin
andymartinwork@gmail.com

Proposal Summary

Many evolutionary programming techniques currently use mutation and a fitness function to select a specific piece of data or function. This proposal outlines a method of creating new types of agent through the use of mutation and a challenging environment as a fitness function. A genetic instruction set will unpack to create the functions of the agent. Asexual reproduction of the agent will mutate the genetic instruction set through genetic copying errors. A result of the mutation will be modification, removal or new function for the child agent. This method does not reach the complexity of modelling biological cells and DNA. It involves a more realistic simulation of the evolutionary process rather than evolving a bit string or function designed for a specific task.

Statement of the Problem

What is intelligence? This is a question that has been asked for thousands of years and yet there is still no clearly defined answer. Most people agree on one aspect of it, however; humans are intelligent.
Despite over 40 years of prediction that a machine with a human level of intelligence is "just a generation away" (Minsky 1967, quoted in Crevier 1993 *AI: The Tumultuous Search for Artificial Intelligence*, p. 109), scientists have still failed to achieve this goal. Some have argued that simulating logic was a simple aspect of our minds and the harder aspects to reproduce are modules of our brain that we take for granted, such as vision (David Marr) and language (Noam Chomsky). Evolutionary psychologists argue that due to a combination of these individual modules that were evolved for specific purposes, intelligence is more of a side effect than an end goal.
Ultimately, building an intelligent mind is a very complicated task. This research proposal may identify a different way of producing more complex programs that lead to intelligence.

Aim

•To create a stable population of agents
•To observe any changes in an agent's useful functions by attempting to mutate an agent's runtime code.

Objectives

A good standard of programming ability must be gained in LISP. LISP may be the best language to write this program, due to its support of dynamically loaded methods for instantiation of genetic code and its ability to re-write its own source code.

A simple agent must be written with the ability to eat, reproduce and die. It must die if it does not acquire enough food or if a certain amount of time has passed.

One or many environments must be written which will provide tests for the agents to prove or disprove their ability to mutate meaningfully.

Genetic language limitations

By using a language suited for writing common computer programs, it will most likely be the case that a vast majority of combinations of operators and variables will cause the agent to malfunction. To reduce the chances of this, the genetic code will be formed of blocks of several statements which can easily be combined to produce behaviour suited to the environment.

Some changes to standard coding practice will be made to ensure an agent can modify itself in a more efficient manner:
•Variables will be named in a sequential manner and be globally accessible inside the agent
•Code blocks that access variables will automatically handle types
•Information about the environment must be accessible to the agent
•The path of execution for an agent must be easily modified
•Methods to detect infinite loops will be required

As a great deal of genetic code produced by mutation will have no useful function (i.e. it will be "junk" DNA), it will be necessary to read only certain parts of the code. In biological DNA, start and stop identifiers called codons are used to mark the readable sections (*Flexibility of the genetic code with respect to DNA structure*). In this proposal a LISP function and enclosing parenthesis will be used.

Mutation rates

Mutation in this context is the change in a child agent's genetic instruction set during reproduction which leads to a modification of that agent's runtime functions.
A mutation rate that is too high will lead to a more diverse, but potentially less fit population due to continued change even after an optimal state has been achieved. A mutation rate that is too low will mean desired changes in agents will take a very long time to occur and may not be recorded in experiment.

To find the optimal mutation rate, a self-adaptive method will be used.
Adaptive methods are used for genetic algorithms and as the paper "*Intelligent Mutation Rate Control in Canonical Genetic Algorithms*" shows, may result in a more efficient convergence to a desired state. This occurs by starting with a higher mutation rate, or learning rate, which gradually moves to a very low mutation rate based on the distance to the fitness value.

Genetic algorithms use a global function to control mutation and a global mutation rate, however bacteria vary their rates by altering "genes coding for DNA repair enzymes and for proteins that assure accuracy of DNA replications".
The proposal in the case of agents in this experiment is to alter the accuracy of gene replication; repairing genes are out of scope.

An example from the biological world of optimum mutation rates are from the paper "Evolution of mutation rates in bacteria":

> *"for Escherichia coli K-12, the rate of deleterious mutations per genome replication is, at least,* $2-8\times10\wedge-4$ *, while that of beneficial mutations is, at least,* $2\times10\wedge-9$ *."*

Although there is a $10\wedge5$ smaller likelihood of this bacterium receiving a beneficial mutation, its genetic structure is more complicated than the genetics suggested in this research proposal.
Its genetic complexity is revealed in the Pubmed abstract "The complete genome sequence of Escherichia coli K-12":

> *"Escherichia coli K-12 contains a 4,639,221-base pair sequence and has 4288 protein-coding genes annotated, 38 percent have no attributed function."*

A small instruction set is aimed at raising the chances of both beneficial and deleterious mutation in order to allow the experiment to fit into available computer hardware and time constraints.

Program execution considerations

A limitation on the scope of the research will be the size of computer memory.
Computers currently have a memory size measured in Gigabytes. This limits the size of both the agent and of the maximum number of agents running simultaneously.

A suggested size for an agent would be 1024 bytes. This will contain a genetic instruction set which is possibly compressed, an agent's running variables and its running methods. It may also contain redundant genetic material as a result of copying mutation. 1024 bytes was specifically chosen to ensure agents will fit neatly into blocks of memory.

As agents in the proposed simulation will have 1024 bytes of memory, only one million agents will fit into a block of one gigabyte of memory. It may be necessary to change the size of the agent or the number of agents based on results of experimentation.

CPU requirements will also be a consideration. Consider a 1 GHz single-core processor. One CPU cycle will take 0.000000001 seconds to execute. If an agent's running functions take 1,000 cycles to complete (including overhead), 1,000,000 agents will complete their operations in the space of one second. This figure is an estimate and will need to be confirmed in a set of performance tests.

Due to the number of agents running in planned simulations, simultaneous execution is not possible. A queuing system for the agents would need to be created in which each agent has its functions executed in turn.
The nature of the environment means a crude interaction is necessary between the agents, in terms of adjacent food and reproduction and moving.
A paper *"Evolution in Asynchronous Cellular Automata"* suggests an interesting way to schedule processing of cellular automata by giving each cell a state and executing functions in each cell when its neighbourhood enters a "ready" state. Ultimately, tests will need to be run to find out if this is a feasible alternative to simultaneous execution.

Structure of proposed agents

```
agent {
list genes //contains encoded form of all functions and variables below
int age
int[][] position[][]
int food

function age {} //call eat, increment age and call reproduce or die depending on variables
function eat {} //increment food, if food is adjacent, and remove food from environment grid
function reproduce {} //spawn a copy on an adjacent cell. Mutation occurs during reproduction
function die {} //terminates reference to agent and changes environment grid cell to food

}
```

Agent fitness test

– "Volcanic Vent" test

A two-dimensional grid of cells represents the environment. On a graded scale from left to right of the environment its hostility to agents increases, causing the agents to die at a faster rate. On the same graded scale, the food is replaced at a greater rate allowing agents to reproduce at a faster rate and a smaller chance of starvation.
Agents will include in their code a protective layer that defends against the hostility, however this layer takes food to maintain.

A possible result in the experiment would be for the protective layer to be removed in the areas where it is not required, and additional layers to be added in areas where hostility demands it.

– "Growing Eyes" test

A three-dimensional grid of cells represents the environment. All agents will be confined to a two-dimensional grid and will have a constant rate of food replenishment in order to maintain a steady population. Cells on a second two-dimensional grid directly above the first will be occupied for a specific amount of time, during which the equivalent cell below on the first grid will have a value representing "light" flagged. When this time has elapsed, the agent on the first grid will be terminated.
Agents will have the ability to move based on a trigger.

A possible result in the experiment would be that the move function would factor in the light flag to avoid the agent's termination.

Significance of proposed research

Although evolving agents with the level of intelligence of "bacteria" have little significance in itself, the use of evolution to increase complexity may have a far larger importance. Desirable features in agents may be developed by changing an environment gradually instead of engineering those features directly. This research will most likely not have any current benefits on the discipline, as the resource requirements of more complex versions of this program will require hardware that is orders of magnitude of greater power than that of today's.

Bibliography

Flexibility of the genetic code with respect to DNA structure – Pierre-François Baisnée, Pierre Baldi, Søren Brunak, Anders Grom Pedersen

Intelligent Mutation Rate Control in Canonical Genetic Algorithms – Thomas Bäck, Martin Schütz

Evolution of mutation rates in bacteria - Erick Denamur, Ivan Matic

http://www.ncbi.nlm.nih.gov/pubmed/9278503 - The complete genome sequence of Escherichia coli K-12

Evolution in Asynchronous Cellular Automata – Chrystopher L. Nehaniv

Research Schedule

•Literature Review of problem domain. A thorough read through subject areas of Autonomous agents, Genetic algorithms and Biological genetics.
•Learn programming language. LISP will be required for this research.
•Create a basic agent and basic environment. Basic agent will have a structure as stated in "Structure of proposed agent" and basic environment will resemble a featureless cellular automata grid with a constant rate of food replacement and asynchronous execution.
•Run performance tests. Tests include memory fill and CPU time per generation.
•Write genetic code interfaces
•Write mutation generator
•Create "Volcanic Vent" and then "Growing Eyes" tests
•Review findings after runs