Universal Turing Machine Search

EECS 600: Ruby on Rails

Steven Dee
Justin Gray
Joshua Lee
Neil Sandberg

{sxd98,jxg274,jcl26,nls13}@cwru.edu

March 2, 2009

1 Abstract

We propose to investigate Darwin's Theory of Natural Selection, in abstract, by analyzing the results of a Genetic Algorithm optimization on a population of Turing Machines. Using the Turing machines as analogs for simple organisms, the goal of the research is to perform an optimization where the result is a population which contains some Universal Turing Machines. We take Universal Turing Machines to represent more complex organisms on an evolutionary path. We will include a web based visualization environment, written in Ruby on Rails, to help visualize the results of the optimizations.

2 Introduction

In 1861 Charles Darwin published his theory of natural selection as a means by which living creatures evolved to best meet the challenges presented to them in their environment. (Darwin 1861) According to Darwin, species undergo changes because certain members of a population posses traits which allow them to reproduce more successfully than others. Those organisms offspring retain the advantageous traits and hence continue to reproduce more successfully themselves. It is an interesting thought experiment to consider Darwin's theory in the extreme, where life exists as very simple organisms and progressively evolves into more complex ones eventually resulting in mammals and then primates and then humans. When examined in the extreme, Darwin's theory seems almost impossible.

We have developed a method to examine this extreme view of Darwin's theory, in an abstract manner, by applying a global optimization process to

Turing Machines. Turing Machines, initially conceived as a thought experiment by Alan Turing in 1937, "are simple abstract computational devices intended to help investigate the extent and limitations of what can be computed". (Barker-Plummer 2004) These simple machines can be constructed so that they are capable of computations that range in complexity from essentially useless to practically unlimited. In their most complex form, Turing Machines are given a special designation: Universal Turing Machines (UTM). A UTM can reproduce the output of any other Turing Machine (even other UTM's), given the same input. We use employ special capability of a UTM as an analogue to the complexity that exists in modern living organisms. By using Turing Machines as the subjects of a global optimization, and carefully constructing the function to evaluate the fitness of each Turing Machine, we will attempt to determine if an optimization can be created where the natural result is a UTM.

There are many methods of global optimization, but the most fitting method for this research is genetic algorithm optimization. The procedure is designed to mimic the evolutionary process described by Darwin's Theory. In a genetic algorithm, potential solutions are encoded as strings, the fittest of which survive to produce more solutions. The key to a successful optimization using the Genetic Algorithm is the selection of the fitness function with which to evaluate the subject of the optimization. Our research will focus on employing a series of fitness evaluation methods and investigating which ones will produce the most interesting Turing Machines at the end of the optimization. This search process is intended to present a simple simulation of the more complex process of evolution in nature.

3 Project Plan

In order to complete the project, we will need to construct a genetic model for Turing Machines. For this, we will need an encoding of Turing Machines in some serial format; a method of scoring Turing Machines that favors universality; and a genetic algorithm that fits the problem. In order to actually see our results, we will also need some way of visualizing them.

3.1 Turing Machine Encoding

We need to come up with a manner of serially encoding Turing Machines such that they can be sensibly modified by a genetic algorithm—essentially, a Turing Machine genome.

3.2 Turing Machine Scoring

We must engineer a way of scoring arbitrary Turing Machines such that those closer to "universality" score higher than those farther away.

We can make the following conjecture: a Turing Machine that is closer to being universal will have more complex (in the Kolmogorov sense) output

than one that is farther from being universal. Then complexity approximates universality, and we should give more weight to Turing Machines that produce complicated-looking output, regardless of what it is.

Alternatively, we can make use of the definition of a UTM given by David Barker-Plummer (Barker-Plummer 2004), where we feed the encoding of a randomly generated Turing Machine along with an randomly generated input to a candidate Turing Machine in the population. We score then score the candidate Turing Machine on how closely its output approximates the output of the randomly generated one.

3.3 Results Visualization

As part of our project, we will produce a visualization system for the results of our process. Specifically, we will use the Ruby on Rails web framework to create a web application that displays Turing Machines generated by our algorithm, including genome, structure, and sample output.

4 Project Timeline

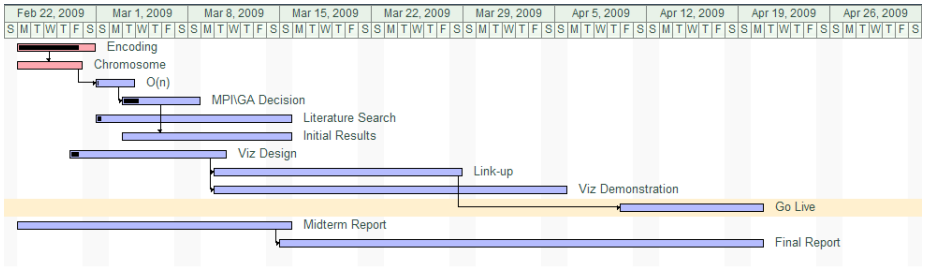

Figure 1: Project timeline of major milestones

Figure 1 shows a detailed timeline for the completion of the project. The major milestones for all aspects of the project present, including necessary work for the creation of an optimization scheme to search for UTMs as well as the creation of a web based visualization tool to facilitate the examination of the project results. The key to the task names used in Figure 1 is presented below:

- **Encoding**: Definition of the encoding method for Turing Machines completed
- **Chromosome**: Implementation of the chromosome class for the AI4R Genetic Algorithm package completed.
- **O(n)**: Computational cost for fitness evaluation of a Turing Machine is determined.

- **MPI/GA Decision**: Decision made regarding necessity of use for MPI along with AI4R Genetic Algorithm to reduce computation time.
- **Literature Search**: Literature search to generate a list of known UTMs
- **Initial Results**: Initial Turing Machine Optimization run completed. Data analyzed to ensure the viability of optimization scheme
- **Viz Design**: Visualization web-app design complete
- **Link-up**: Link between genetic algorithm generation data and visualization database completed.
- **Viz Demonstration**: Ruby on Rails visualization application implementation completed.
- **Go Live**: Ruby on Rails visualization application goes live with link to genetic algorithm running perpetually.
- **Midterm Report**: Midterm Report Delivery
- **Final Report**: Final Report Delivery
- **Presentation**: Final Presentation

5 Organizational Structure

5.1 Project division

A first priority will be to define a standard format for representing a Turing Machine; this format will be used pervasively in the entire application. From there, several independent modules fall naturally out of the problem specification:

- Execution: Given a TM and its input, returning successive states of the tape at points in the future. (Josh Lee)
- Evaluation: Determining a heuristic for universality, ranking TMs, and producing stochastic test data. (Justin Gray, Steven Dee)
- Breeding and mutating: The GA's selection and combination processes. (Neil Sandberg)
- Web based visualization tool, using Ruby on Rails. (Justin Gray, Neil Sandberg)

It is our expectation that, since these modules have well-defined interfaces and can be tested independently of each other, we will be able to work on them individually or in pairs, depending on the size of the task. The responsible parties are listed next to each module

5.2 Communication

We will use a mailing list to coordinate communication for the group. Justin Gray will serve as the point of contact for the group as a whole. Github will be used as a source repository to enable distributed collaboration.

6 Risk Analysis

Our hypothesis is that applying our selected fitness evaluation methods to a set of Turing Machines will lead a genetic algorithm optimization to an optimum that is a set of UTMs. It is possible, using a single fitness evaluation, that we could get no UTMs. In that case, the data will have shown that our particular fitness evaluation does not lead to universality. This however would not be a wholesale disproval of the research method. To deal with this possibility, we will employee a set of fitness evaluation methods and compare their results. This way, a slightly broader dataset can be generated. Though both methods still can not conclusively disprove the research method, together they provide a much stronger conclusion in either direction.

At this point, the amount of time necessary for the optimizations to run is unknown. We assume that the amount of time it takes to evaluate a single member of the candidate population is trivial. If this does not turn out to be the case, then it is possible that the optimization could benefit greatly from the usage of parallel evaluation of population members. We have identified an implementation of the Message Passing Interface (MPI) for Ruby which could be used to implement such a parallel scheme. By March 8th, we will have evaluated the computational requirements for fitness evaluation and if necessary can begin work on implementing a parallel optimization algorithm.

Even if the computational requirements for evaluation of a candidate Turing Machine are modest, the optimization algorithm could move toward an optimum slowly. Genetic Algorithms are known for their slow convergence rates. We have provided ample time to allow for this potential bottleneck, by starting a test of our optimizations on March 3rd. That will provide plenty of time for a slow optimization to run, leaving time to analyze the results.

References

Barker-Plummer, D. (2004), 'Turing machines'. Stanford Encyclopedia of Philosophy.
URL: *http://plato.stanford.edu/entries/turing-machine/*

Darwin, C. R. (1861), *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*, 3 edn.

Fierense, S. & Ferret, J. (2009), *Genetics Algorithms in Ruby :: ai4r*.
URL: *http://ai4r.rubyforge.org/geneticAlgorithms.html*