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 (TM). TMs, 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, TMs are given a special designation: Universal Turing Machines (UTM). A UTM can reproduce the output of any other TM (even other UTM's), given the same input. We employ special capability of a UTM as an analogue to the complexity that exists in modern living organisms. By using TMs as the subjects of a global optimization, and carefully constructing the function to evaluate the fitness of each TM, an evaluation of the possibility that "complex" TMs can evolve from effectively random "simple" TMs is performed.

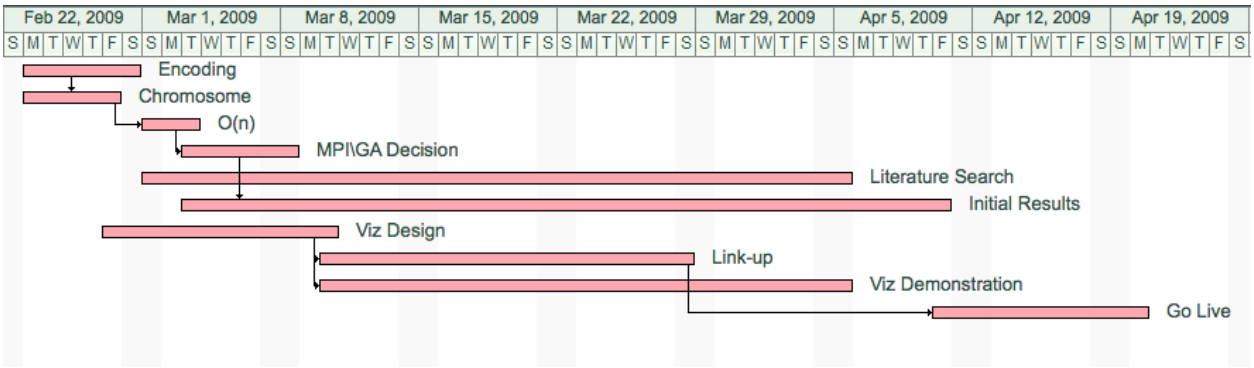

Figure 1: Project timeline of major milestones

3 Project Timeline

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 TMs
- **Chromosome**: Implementation of the chromosome class for the GA
- **O(n)**: Computational cost for fitness evaluation of a TM is determined.
- **MPI/GA Decision**: Decision made regarding necessity of use for MPI along with Genetic Algorithm to reduce computation time.
- **Literature Search**: Literature search to generate a list of known UTMs
- **Initial Results**: Initial TM optimization run completed. Data analyzed to ensure the viability of optimization scheme
- **Viz Design**: Visualization web-app design complete
- **Link-up**: Link between GA generation data and visualization database completed.
- **Viz Demonstration**: Ruby on Rails visualization application implementation completed.

4 Organizational Structure

4.1 Project division

The project was devided into two main parts: The genetic algorithm optimization and the results visualization. The work load was split evenly between team members, but each section was managed by lead. Josh Lee led on the genetic algorithm implementation and Neil Sandberg led the web visualization implementation. The genetic algorithm implementation required the larger development efford, and Josh Lee and Steven Dee focused their efforts on this development, with algorithm design support provided by Justin Gray. Neil developed the initial web visualation tool and Justin Gray contributed to the final data representation design and implementation.

Each of the two parts was developed in parallel, using a database as the data bridge between them. By using a simple database design that was fixed early on, the task of integration of the two parts was greatly simplified.

All work was performed collaboratively using a github source repository. The project is stored in an open source repository at http://github.com/mrdomino/utm/. The repository was used to store all files regarding the implementation of the genetic algorithm and the web visualization tool.

5 Genetic Algorithm Design

Genetic algorithm optimization is designed to mimic the evolutionary process described by Darwin's Theory. In this genetic algorithm, potential solutions are encoded as strings, the fittest of which survive to produce more solutions in later generations. 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. However, determining the fitness test was not the only step required to execute the optimization. A method of encoding a TM into a "genome" was required along with some method of reproduction between parent TMs.

5.1 Turing Machine Encoding

The encoding translates a binary string into an actual TM. Binary string encoding was chosen for its simplicity and well documented use in genetic algorithms. To define the encoding scheme, first it was necessary to define a set of constants for the TM population:

- NUM_STATES: Total number of states represented in each state table
- GENE_LENGTH: The number of bits being used to represent a single entry in the state table

The size of GENE_LENGTH is required to be at least large enough to represent NUM_STATES while reserving two bits at the end for the **write** bit and **movement** bit(left = 0, right = 1). However, it is allowed for there to be many more bits than necessary to accommodate future growth in NUM_STATES without changing the alignment in all previous TMs. To handle this issue, the modulus operator was employed to ensure that all possible state values fall within a valid range.

current state	read bit	next state	write bit	movement
1	0	11	1	1
1	1	10	1	0
2	0	10	0	1
2	1	10	0	1
3	0	01	0	1
3	1	01	1	1

Table 1: State transition table for notional 3-state TM where the next state has been represented in an appropriate length binary string

Take the example encoding: '111110101001100101000110', with NUM_STATES=3 and GENE_LENGTH=4. This is broken up into segments which are GENE_LENGTH bits long: 1111,1010,1001,1001,0100,0110. Each gene segment then represents a single entry in a state transition table. The resulting state transition table can be seen in Table 1.

It should be noted that all TM using this encoding assume that the 0 state is the halt state. It is not required that any TM utilize the halt state at all. The example TM above did not use the halt state.

5.2 Fitness Evaluation

The function of the fitness evaluation in a Genetic Algorithm is to provide the means of scoring one population candidate against another. The particular formulation that is chosen for a fitness evaluation will effect the entire outcome of any optimization. For this research, the purpose is to investigate the likelihood that an evolutionary process can produce very "complex" organisms from "simple" ones. Taken TMs as analogs for organisms, the best possible result we can hope for is a population of UTM's. So the fitness function employed must grade TMs that are most like a UTM higher than ones that are less like a UTM.

There is not simple test for that can be applied to a TM to determine if it is in fact a UTM. If there were, you would simply apply that test to every member of a population and grade them accordingly. Lacking such a test, it is necessary to establish certain measurable qualities of a UTM that could be tested for on a candidate. The fundamental problem with this approach is that when a specific candidate has been shown to share a lot of the qualities that you have associated with a UTM, there is no single logical path that can extend that information to show that the candidate is actually a UTM (that's why there is no test!).

Instead of testing directly for qualities of candidate TMs that are similar to those of a UTM, a more simple fitness evaluation was used, which follows more directly with the stated hypothesis. TMs are scored based on the complexity of their output. This complexity is measured by applying a compression algorithm to the output string of the turing machine, and then using the resulting size of the compressed data as the fitness of the TM. The DEFLATE compression algorithm was selected because it is a lossless compression algorithm and is part of the ruby standard library.

Using this method will provide higher fitness scores for more random outputs. However, a side effect of using the DEFLATE algorithm is that a small size advantage is given to simple repeating patterns. For example, take the following three cases:

- 000000000000000000000000 — fitness: 11
- 101010101010101010101010 — fitness: 12
- 101101101001010110101010 0 — fitness: 20

The first case is as simple as it gets, just one character. The second case is a simple repeating pattern of ”01”. The second case holds a small advantage over the first, but compared to the score of the very random third case (12 vs 20) the slight increase in not significant enough to invalidate the fitness evaluation method.

Since the output of a TM is dependent on the input given to it, the outcome of this fitness evaluation is highly dependent on the input given to each candidate TM as well. Three interesting input types were evaluated:

1. Fixed input string for all generations
2. Randomly generated input string for each generation
3. Input string with some mutation between generations

The first option tests the outcome of an optimization where each generation is being given effectively the same exact test as all the others. TMs which are good at making complex output for this specific input will thrive. The second option basically generated a random new test for each generation, and only TMs that can produce complex output for arbitrary input will thrive. These TMs could be considered very adaptable. The third option presents a hybrid approach to the previous two; here each successive generation is operating on an input string which is similar to the last (how similar depends on the mutation rate) but not identical. Over a large number of generations however the input string is effectively appears random. For this research, the when using the third input string option, a mutation rate of 15% was used.

5.3 Genome Mutation

The mutation operator in a GA serves to introduce new genes to the population. Without that injection of new genes the only possible results would have to consist of some combination of the genes that were created at random as part of the initial population. The mutation algorithm applied here used a bit swapping method where a certain percentage of bits in the encoding are swapped with their neighboring bits. Using this method both the probability that a genome will be mutated and the frequency of bit swaps along the encoding can be varied to adjust the performance of the genetic algorithm optimization. For this work, the probability of mutation was set to 30% and the frequency of bit swaps was 10%.

Keeping these parameters set to higher values ensures a constant and high rate of introduction of new genes into the population. This much churning of the gene pool also helps to prevent too much homogeneity from developing in the gene pool.

5.4 Parent Selection and Breeding

Between each generation a GA selects a subset of the current population for breeding to produce the next generation of candidates. This research utilizes the competition selection method. The competition selection method was chosen for the flexibility it provides in tuning GA to preserve population diversity. The competition method selects parents for breeding by creating a competition with COMPETITION_SIZE contestants. The contestants are selected at random from the population and the one with the highest fitness wins and becomes a parent. By making COMPETITION_SIZE smaller the probability that a weaker contestant will win increases, thus preserving diversity by allowing weaker genomes to survive to the next generation. For this research COMPETITION_SIZE was set to 2, ensuring maximum diversity.

6 Web Visualization Tool Design

The results visualization system provides a graphical summary of the progression of an optimziation. It was written using Ruby On Rails and takes advantage of two graphical plotting tools: gcharts and graphviz. Gcharts is used to generate the graphcial plots of relevant optimization metrics from different optimizations. The graphviz package is used to render images of the

6.1 Optimization Metrics

The progress of the optimziation is charachertized on a per generation bassis. Relevant metrics include:

- Average fitness
- Maximum fitness

For a good optimization one would expect both the maximum fitness and the average fitness to increase from one generation to the next. However, the data represnetation provides valuable information regarding the relative values of the maximum and the average fitness. If the average fitness begins to approach a value close to the maximum fitness, this is an indication that the population is becoming too homogeneous and it is unreasonable to expect further optimization to result in large fitness gains. Since the optimization was

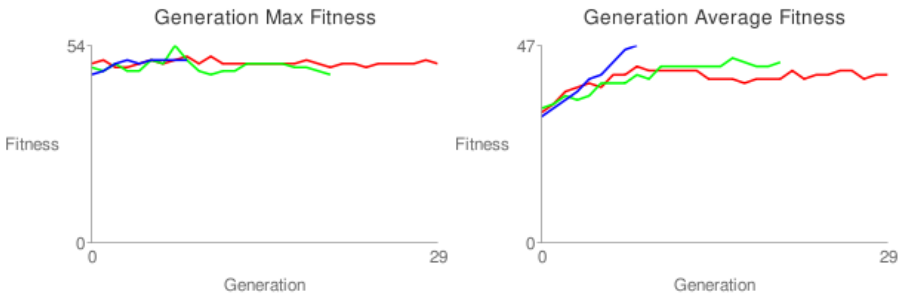

Figure 2: Example metric graphs for maximum and average fitness.

sub-devided into three separate optimization runs to test the three different fitness testing methods, the visualization tool also provides graphical representation of each of the three sub-data sets simultaneously. A sample set of graphs is shown in Figure 2 This representation allows a quick comparison between the effectiveness of the three different fitness evaluation methods.

6.2 TM Representation

The vizualization of individual TMs provides a valuable tool for their inspection. There are litterally thousands of TMs created over the course of an optimziation, and graphviz package allows or the creation of a state transition diagram for quick visual inspection of particular TMs. Figure 3 is the state transition

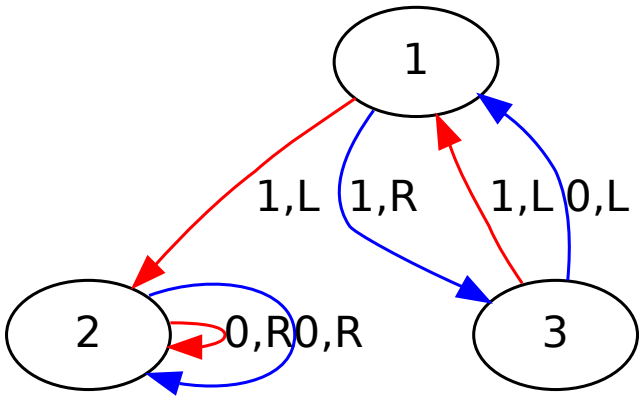

Figure 3: State transition diagram for notional 3-state TM. Edge color indicates the bit being read from the tape (red=0,blue=1). Edge label indicates the write bit and movement direction for a given transition.

diagram for the example TM in Table 1. This visual representation is much easier to interperate than the state transition table.

7 Known Universal Turing Machines

Alan Turing published his discovery of the first UTM in his original work on TM's. (Turing 1936) In 1956 Claude Shannon proved that it was possible to have a UTM with only two-symbols and that it was possible to exchange the number of states with the number of symbols. (Shannon 1956) His research sparked the search for the smallest possible UTM, in terms of the number of states and symbols needed to describe it.

In 1967 Marvin Minsky found a very small UTM that needed only four symbols and seven states. (Minsky 1967) in 1985 Stephen Wolfram found a smaller UTM with two states and 5 symbols. (Wolfram 2002) His work, supported by Mathew Cook found a series of small UTMs, culminating in the discovery of what is currently believed to be the smallest possile UTM. Wolfram discovered a two state, three symbol TM which he believed to be a UTM. He could not, however, prove its universality. An undergraduate student, Alex Smith did prove its universality in 2007. Figure 4 shows the graphical representation that Wolfram uses to describe a TM. (Wolfram 2007)

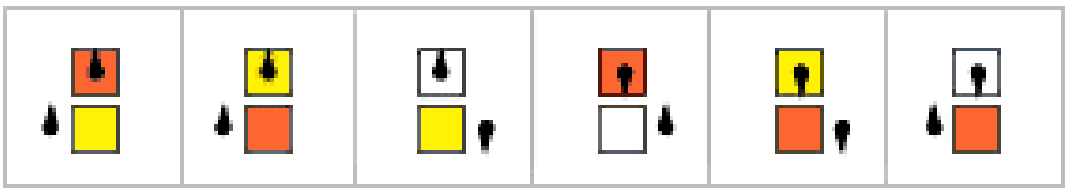

Figure 4: The wolfram representation of the 2,3 UTM. It is the smallest possible UTM. The arrows indicate state and the colors indicate the tape symbol. Tape movement direction is shown by the left/right location of the "to state"

Though the majority of past research has focused on the discovery of ever smaller UTMs, this work trends in the oposite direction. Organic creatues use dna to encode their description. Feverati and Musso, in

2008, performed research using a GA to find complex TMs. Their goal was to use their model to investigate the usefullness of non-coding regions in dna encodings with regard to the creation of complex organisms. Their conclusion was that large non-coding regions, in this case represented by inaccessible states in the state-transition table, provided an evolutionary benifit.(Feverati & Musso 2007) This research implies that using encodings that provide a large number of possible states

8 Results

8.1 O(n)

The computational expense of evaluating the fitness of a TM will vary depending on the particular implementation of the fitness algorithm. The use of an interpreted programming language also provides a challenge with regard to computational expense. For this research it was necessary to evaluate the computational expense for this particular fitness algorithm and implementation in ruby.

For each of the three fitness evaluation algorithms the computational expense of a single fitness evaluation was less than .2 seconds. At this level of computational expense it was not necessary to consider parallel execution implementations for the GA. However, if other fitness algorithms were used which increased the computational cost of a single evaluation to approximately 1 second or longer it would be beneficial to consider parallel computing. To help keep the computational cost down, the length of the output tape for each TM was limited to a maximum length of 5000 bits. When the tape reaches that length the TM is forced to halt. This restriction is somewhat artificial, but it helps to keep the computational cost down. It also prevents TM with very long, but very simple outputs from scoring much higher than more random but shorter outputs. By setting the maximum length

GAs lend themselves to parallel computation because for a given population, each member of the population can be evaluated independently of all the other. This allows GAs to be constructed using Message Passing Interface (MPI) which can make use of multiprocessor machines and network based computational clusters.

8.2 Optimization Results

Each of the three types of fitness algorithms were tested by running three separate optimizations. All of the optimizations were run with the same control parameters:

- NUM_STATES: 64
- GENE_LENGTH: 2
- population size: 100
- generations: 100

The generational history for each optimization is presented in Figure 5. The maximum fitness overall was 55. The initial average fitness of the first generation was nearly 0, but the initial maximum fitness was 48. The difference of only 7 points between the initial and the final maximum fitness indicates that there was not a significant increase in output complexity discovered over the course of the optimization. The deflate compression algorithm used to compress the output strings displays a small preference for heavily repeating patterns, and it is possible that this small difference reflects this tendency. However, it is also possible that these optimization results represent the maximum complexity achievable in a 5000 bit output tape.

When comparing the results between each of the three separate fitness evaluation techniques, from Figure 5 it is obvious that none of the three methods hold a significant advantage in finding a more complex machine. In all three cases, by the 50th generation the optimization rate slowed to a low enough speed that further optimization was not necessary. In fact, in the case of the random input tape case, the red line, the overall maximum fitness was actually discovered very early on and rediscovered much later on.

One stated goal of the optimization was to maintain as much population diversity as possible. The average fitness data history provides an indication of the relative diversity and how it changed over course

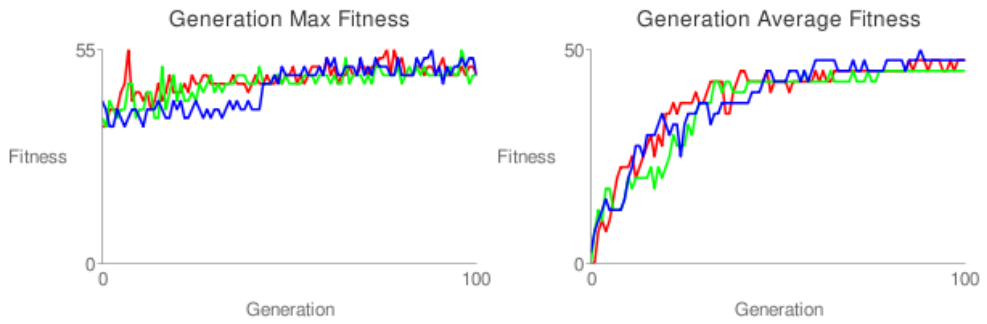

Figure 5: Final optimization results for three optimizations. Red: random input tape, Green: constant input tape, Blue: Slow drift input tape

of the optimization. As the generations progressed the average fitness starts out near zero. It climbs very quickly as the weakest of the population die out first. The average fitness begins to approach the maximum fitness as the generations progress because the population begins to converge on the strongest candidates. Once the population reaches a certain amount of homogeneity the optimization it is unlikely that any new members will emerge with a strong advantage over the others because the gene pool is too small. At this point, only mutation could provide the necessary injection of new genetic material. Because of the large potential non-coding regions of the 64 state encodings, it is possible that a mutation would turn on a dormant part of the genome and allow for a strong advantage, but in this optimization all three methods had approximately 50 generations to allow this to happen and it did not.

Since none of the three variations in fitness algorithms produced significantly improved results, it is fair to conclude that none of them provides a significantly different result than any of the others. It is interesting that all three input methods result in similar TM whose output reaches a similar amount of complexity.

One notable aspect of the results is the oscillating results. This effect is most notable in the red data, the random input tape method. This optimization reached a maximum early on and took a few generations to refind a TM with similarly complex output. This effect is notable because it illuminated a potential weakness in the GA used. This GA does not ensure that the strongest population member of a generation survives to the next generation. It is highly likely that it will survive, but in the event that a single population candidate becomes significantly stronger in a single generation, it is possible despite its overwhelming strength it will not survive to the next generation. If it does not survive, its genetic material is lost and would need to be rediscovered later. This property is mostly a consequence of the selection method used, competition with only two contestants. This method was used to help ensure maximum diversity, but it also has the side effect of increasing the probability that a single strong population member would not get to enter a competition and so it would never win one and never reproduce. This problem could be remedied by either increasing the size of competitions or by switching to a roulette selection method. Either solution would greatly increase the chance that the best candidate will survive.

Neither of these methods will absolutely guarantee that the strongest candidates survive. It is, of course, possible to simply always pass the strongest population member to the next generation. However, doing so moves the algorithm away from mimicking nature as closely as possible. In nature, there is no guarantee that even very strong organism will have a chance to mate. Any number of things can happen to prevent it to happen, like a pretator or some natural distator. Future research could investigate the efficacy of trying to always preserve the strongest candidates, either by increasing their chance of survival or by simply making it happen.

9 Conclusion

The result of all three optimizations were basically the same. They each reached a maximum fitness of 55. It is reasonable to conclude that, with the DEFLATE compression used that 55 could be close to the maximum size a 5000 bit output string could compress to. Hence, it is also reasonable to conclude that the GA performed as

expected and, for all three cases, found the most complex possible organism. However, since the relative complexity of the best TMs was not significantly higher than the average complexity of majority of the generations, no direct conclusion can be drawn about the universality of the best candidates. A stronger conclusion, though not an absolute one, could be drawn if the maximum complexity was significantly higher than the average. Since this was the case, finding a UTM can neither be rulled out or confirmed.

In any event, it seems likely that the 5000 character output limit may have unintensionally limited the efficacy of the otimization. Future work will certainly need to greatly increase the limit on the size of the output tape. However, doing so will increase the compuational cost of each fitness evaluation. If the time required for a single fitness evaluation grows, it would become necessary to reconsider the necessity for parallel computation.

Future research could also include an increasing in the size of alphabet allowed for candidate turing machines. This would require a slight re-design of the encoding method to allow for more than just a single bit to represent the write bit. Making this change might not affect the chances of actually finding a UTM, but it would make it easier to compare the results to known UTMs. There are very few known UTMs with only two symbols in their library, but the list grows signficantly if you allow the libary to increase. Though the comparison would be done during post processing, it would provide a more direct method to evaluate fitness algorithms after the optimziations has completed.

References

Barker-Plummer, D. (2004), 'Turing machines'. Stanford Encyclopedia of Philosophy.

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.

Feverati, G. & Musso, F. (2007), 'An evolutionary model with turing machines'.

Minsky, M. L. (1967), *Computation: finite and infinite machines*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Shannon, C. E. (1956), A universal turing machine with two internal states, *in* 'Automata studies', pub-PRINCETON, pp. 157 – 170.

Turing, A. M. (1936), 'On computable numbers, with an application to the entscheidungsproblem', *Proc. London Math. Soc.* **2**(42), 230–265.

Wolfram, S. (2002), *A New Kind of Science*, Wolfram Media.

Wolfram, S. (2007), 'Wolfram 2,3 turing machine research prize'. http://www.wolframscience.com/prizes/tm23/.