A Model for Recursively Self Improving Programs

Matt Mahoney

Draft v3, Sept. 14, 2008, revised Dec. 17, 2010

mmahoney@cs.fit.edu, matmahoney@yahoo.com

Abstract

We formally define recursively self improving programs and show that such programs exist, but that their algorithmic complexity cannot grow faster than O(log n) given fixed goals.

Background

Vinge [1] predicted that if humans could produce artificial intelligence (AI) systems with greater than human intelligence, then so could those systems, only faster. The resulting positive feedback cycle would result in an intelligence explosion or singularity. The Singularity Institute for Artificial Intelligence [2] is investigating the threat of runaway AI, but the problem is poorly understood; lacking any precedent. Of central importance is an understanding of the limits of recursive self improvement (RSI). If RSI is possible, then it is critical that the initial goals of the first iteration of agents (seed AI) are friendly to humans and that the goals not drift through successive iterations, assuming we cannot control the goals of subsequent generations appearing in rapid succession. This is an extremely difficult problem in itself, because friendliness is not well defined, although attempts have been made [3]. On the other hand, if RSI is not possible, then we are possibly left with an evolutionary process in which self modifying and reproducing agents compete for computing resources with the most successful of these displacing the others. Any goal contrary to rapid reproduction would be unstable. The evolutionary process might be greatly accelerated because it might be based on software (such as self modifying internet worms) or self replicating nanotechnology (gray goo [4]) rather than on DNA.

To talk about RSI, we must first define it. Generally, we wish to improve intelligence, which is itself hard to define. Turing proposed an imitation game [5] as a test for intelligence: the ability of a machine to fool a human into believing it was another human is sufficient (but not necessary) evidence of intelligence. However, the Turing test is not an appropriate test for RSI because it does not describe a test for superhuman intelligence. This seems to be a general problem. We can detect an IQ of 200 in children, but there is no equivalent test for adults. If Alice is smarter than Bob, then how is Bob to test Alice's intelligence? If Alice gives an answer that Bob believes is wrong, then who are we to believe?

Legg and Hutter [6] surveyed a number of common definitions of intelligence and offered a formal definition of universal intelligence which does not depend on human characteristics. An agent's intelligence is defined as the expected reward in environments modeled by programs generated by coin flips in the AIXI model. In the AIXI model [7] an agent and environment are modeled as a pair of interacting Turing machines in which at each step the agent sends a symbol to the (unknown) environment and the environment sends a symbol plus a reward signal to the agent. The goal of the agent is to accumulate the greatest possible reward. A formal definition has the advantage of allowing mathematical analysis. For example, Hutter proved that the optimal strategy of the agent is to guess at each step that the environment is simulated by the shortest program consistent with all of the interaction observed so far. This would solve the general intelligence problem once and for all (for systems that fit the model), except for the fact that the strategy is not computable.

Legg and Hutter's universal intelligence cannot be measured precisely because it would require testing over an infinite number of environments, although it could be approximated by random sampling.

Wang [8] offers a different definition (also after a survey of common meanings): the ability to adapt to an environment given insufficient knowledge and resources. AIXI has insufficient knowledge (none initially) but unlimited space and time. However, Hutter also proved that in the case of space bound l and time bound t, that there is an optimal and computable (although intractable) strategy called AIXI^{tl} that runs in $O(t\,2^l)$ time [7].

What does self-improvement mean?

We must define what it means for a program to have a goal and to improve with respect to achieving that goal. We sometimes speak of a search procedure or optimization process of having a goal of solving a problem, although this is really the goal of the programmer. However, it is still a useful metaphor for understanding some systems. For example, we can think of a linear regression algorithm having the goal of fitting a line to a set of points with minimum error, or a thermostat having the goal of keeping the room at a certain temperature, or a chess program that wants to win the game. We can compare systems according to how well or how quickly they achieve some goal.

We wonder if it is possible for a system to improve itself, for example, for a program to rewrite its own source code to learn faster, or to store more knowledge in a fixed space, without being given any information except its own source code. This is a different problem than learning, where a program gets better at achieving goals as it receives input. An example of a self improving program would be a program that gets better at playing chess by playing games against itself. Another example would be a program with the goal of finding large prime numbers within t steps given t. The program might improve itself by varying its source code and testing whether the changes find larger primes for various t.

Intelligent agents are normally modeled as interactive processes. However, we may use a batch model in which a Turing machine receives all of its input at the beginning, because we are studying *self* improvement, not learning. (We may, of course, have a goal of learning faster, but it is still possible for a non interactive program to experiment on itself using tests that measure rate of learning).

In the following sections, we formally define what it means for a program to have a goal and to improve with respect to this goal. Improvement is with respect to achieving a goal within a time bound t, which also imposes a space restriction, since it limits how much memory can be accessed within time t. We show that there exists infinite sequences of programs that improve in the sense of producing better answers within time bounds, and give an example of a program that generates the next program in the sequence. Finally, we show that these sequences are severely restricted in their ability to gain useful knowledge.

Goals

We define a *goal* as a function G: $\mathbf{N} \rightarrow \mathbf{R}$ mapping natural numbers (or equivalently, strings, or elements of some other countable set) to real numbers. For example, the function PRIME(x) = {x if x is prime, else 0} represents the goal of finding large prime numbers. G is sometimes called a *utility function*.

Fix a universal Turing machine L. Define P(t) to mean the positive natural number encoded by the output of program P with input t running on L after t time steps, or 0 if P has not halted after t steps. We say that *P has goal G at time t* if and only if there exists t' > t such that G(P(t')) > G(P(t)) and for all t' > t, G(P(t')) ≥ G(P(t)). In other words, running P longer will eventually produce a better result and never produce a worse result afterwards.

Definition: *P has goal G* if there exists a constant C such that P has goal G for all t > C.

Example 1: let G = PRIME and let P = {3 if t < 100, else 5} complete in 50 steps. Then P has goal PRIME for t < 100, but does not have this goal thereafter.

Example 2: let P = {t} complete after t/2 steps. Then P has goal PRIME at time t = 6, but not at t = 7.

Example 3: let P = {n: n is the largest prime number less than $t^{1/3}$} complete in less than t steps for all t > 1000. Then P has goal PRIME (for all t).

If P has goal G, then G(P(t)) is a monotonically increasing function of t with no maximum for t > C. More time always helps. We define a goal this way because otherwise once we find a maximum G(P(t)) we could not say if P is "trying" to find a higher value. We can only measure results, not intent.

Improvement

Definition: *Q improves on P with respect to goal G* if and only if all of the following conditions are met:

- P has goal G.
- Q has goal G.
- There exists t such that G(Q(t)) > G(P(t)).
- There does not exist t' > t such that G(Q(t')) < G(P(t')).

In other words, at some point Q produces a better result than P in the same amount of time and never produces a worse result given more time. The last condition is needed because otherwise it would be possible to construct P and Q such that both improve on the other.

Example 4: let Q = {n: n is the largest prime number less than $t^{1/3} + 1$} complete in less than t steps for all t > 2000. Then Q improves on P from example 3.

Recursive Self Improvement

Define an *improving sequence with respect to G* as an infinite sequence of programs P_1, P_2, P_3,... such that for all i > 0, P_{i+1} improves on P_i with respect to goal G.

Example 5: Let G be the identity goal: G(x) = x (the goal of finding big numbers). Let $P_n(t)$ = {t + n} complete in less than C + D log tn steps for some positive constants C and D. Then this is an improving sequence with respect to G. It is also computable using binary string representations of numbers because addition and output take *O*(log n) steps for numbers n.

Up to this point, we have considered only programs that output natural numbers. We now consider programs that can output other programs. We have previously defined programs P(t) that accept t > 0. Without loss of generality, and without changing our previous definitions, we extend our definitions to allow the value -1 to be accepted as input, such that P(-1) outputs a (suitably encoded) program.

Definition: P_1 is a *recursively self improving (RSI) program with respect to G* if and only if $P_i(-1) = P_{i+1}$ for all i > 0 and the sequence P_i, i = 1, 2, 3... is an improving sequence with respect to G.

Example 6: Let L to be the C programming language, with input and output expressed as decimal strings. Let G be the identity goal. Then define $P_1(t)$ as the following program:

```
char*p="char*p=%c%s%c;"
"main(){"
  "int t,n=%d;"
  "scanf(%c%cd%c,&t);"
  "t>=0"
```

```
    "?printf(%c%cd%c,t+n)"
    ":printf(p,34,p,34,n+1,34,37,34,34,37,34);}";
main(){
  int t,n=1;
  scanf("%d",&t);
  t>=0
    ?printf("%d",t+n)
    :printf(p,34,p,34,n+1,34,37,34,34,37,34);}
```

(34 and 37 are the ASCII codes of " and % respectively). Successive iterations P_2, P_3,... are written on one line, but differ functionally from P_1 only in that the initialization "n=1" is replaced by "n=2", "n=3", etc. Strictly speaking, this program is not RSI because the addition will eventually overflow. But in principle, RSI programs are possible given unlimited memory to encode arbitrarily large integers.

Bounds on RSI

In example 6, the algorithmic (Kolmogorov) complexity of P_n is $O(\log n)$. This is because the program has constant complexity except for the encoding of n, which can be expressed in $O(\log n)$ digits. In fact, this is true in general.

Theorem: Let P_n be the n'th iteration of a recursively self improving program. Then the algorithmic complexity of P_n is not greater than $O(\log n)$.

Proof: recall that the algorithmic complexity of a string x is the length of the shortest program that outputs x. P_n can be output by a program that takes P_1 and n as input, sets $P = P_1$ and iterates $P \leftarrow P(-1)$ n times. The control program and P_1 have fixed size which does not depend on n. n can be encoded in $O(\log n)$ bits.

We may conclude that if intelligence is a function of knowledge and computing power, then software self modification by itself is not a viable path to self improvement because neither is increased.

Acknowledgments

I thank Pei Wang and Eliezer Yudkowsky for helpful comments on this paper.

References

1. Vernor Vinge, "The Coming Technological Singularity: How to Survive in the Post-Human Era", Whole Earth Review, Winter 1993. http://www-rohan.sdsu.edu/faculty/vinge/misc/singularity.html
2. SIAI, http://www.singinst.org/
3. Eliezer S. Yudkowsky, "Coherent Extrapolated Volition", SIAI, 2004. http://singinst.org/upload/CEV.html
4. http://en.wikipedia.org/wiki/Grey_goo
5. A. M. Turing, "Computing Machinery and Intelligence", Mind, 59:433-460, 1950.
6. Shane Legg and Marcus Hutter (2006), A Formal Measure of Machine Intelligence, *Proc. Annual machine learning conference of Belgium and The Netherlands (Benelearn-2006)*. Ghent, 2006. http://www.vetta.org/documents/ui_benelearn.pdf
7. Marcus Hutter, "A Gentle Introduction to The Universal Algorithmic Agent {AIXI}", in *Artificial General Intelligence*, B. Goertzel and C. Pennachin eds., Springer, 2003. http://www.idsia.ch/~marcus/ai/aixigentle.htm
8. Pei Wang, "On the Working Definition of Intelligence", CRCC Technical Report 94, Indiana University, 1995. http://www.cogsci.indiana.edu/farg/peiwang/papers.html#intelligence