Submitted to the IJCAI 2001 workshop on Empirical AI Seattle, Washington: http://www.ijcai-01.org
WP ref: 01/random/random (March 1, 2001).
URL ref: http://tim.menzies.com/pdf/01random.pdf

Empirically, how Hard is it to Test AI Software?

Tim Menzies
Dept. Electrical Engineering & Computer Engineering, University of British Columbia, Vancouver, Canada;
<tim@menzies.com>

1 Introduction

S^N tests is the upper bound of tests required for a system of N variables with S states per variable. Yet Figure 1 shows that much of the expert systems literature proposes evaluations based on far fewer tests (exception: [Bahill *et al.*, 1995] propose at least one test for every five rules). Such small test sets can never be more than an *approximate test* of a system, but how poor will be that approximation?

This article reviews empirical evidence from the literature and from simulations that suggest that, in the majority of cases, approximate testing will suffice. It is based extensively on material from [Menzies and Cukic, 2001].

Text	N tests	Text	N tests
[Harmon and King, 1983]	4..5	[Buchanan *et al.*, 1983]	≈ 6
[Bobrow *et al.*, 1986]	5..10	[Davies, 1994]	8..10
[Yu *et al.*, 1979]	10	[Caraca-Valente *et al.*, 2000]	< 13
[Menzies, 1998]	40	[Ramsey and Basili, 1989]	50
[Betta *et al.*, 1995]	200		

Figure 1: Number of tests proposed by different authors. Extended from a survey by [Caraca-Valente *et al.*, 2000].

2 Approximate Testing

Approximate testing assumes that:

1. A system can be sampled via a small number of inputs.
2. This small set of inputs might not be chosen with much care.

If our AI software contains pathways tangled like spaghetti, then these two assumption are clearly inappropriate. However, if AI software pathways are simple, then a few tests will adequately probe a system and approximate testing is an adequate strategy.

To see when approximate testing might work, consider the following example containing 15 binary variables $a, b \ldots, n, o$. The S^N equation tells us that this system needs $2^{15} = 32768$ tests. However, suppose that system has only one input and one output and pathways look like this:

$$\begin{array}{c} a \to b \to c \to d \to e \\ \mathit{input} \to f \to g \to h \to i \to j \to \mathit{output} \\ k \to l \to m \to n \to o \end{array}$$

Clearly, no tests are required for the top and bottom pathway since they are isolated from system inputs and outputs. Further, only a single test is required to cover the only pathway that connects inputs to outputs. If the paths in most programs look like this example, then software contains

1. A *small number of simple i/o pathways* from inputs to outputs, and
2. *Large unreachable regions*.

If so, then any input, selected at random, that can propagate through to any output will test a significant portion of the usable parts of the system. Hence:

- A small number of such inputs would hence be sufficient to test a system; and
- We can endorse approximate testing.

3 Empirical Support for Approximate Testing

There is much empirical evidence from the literature that real-world software, including KBS, comprise mostly simple i/o paths and large unreachable regions. For such systems, approximate testing will adequately sample a space.

For example, Bieman & Schultz [Bieman and Schultz, 1992] studied how many sets of inputs are required to exercise all *du-pathways* in a system. A du-path is a link from where a variable is *defined* to where it is *used*. Figure 2 shows their experimental results. At least for the system they studied, in the overwhelming majority of their modules, very few inputs exercised all the du-pathways.

In other work, Colomb [Colomb, 1999] compared the inputs presented to an medical expert system with its internal structure. Based on the number variables N and their states S, Colomb noted that that system should have $S^N = 10^{14}$ internal states. However, after one year's operation, the inputs to that expert system only exercised 4000 states; i.e. in practice, this system only needed to handle a tiny fraction of the possible states ($4000 \ll 10^{14}$).

Avritzer et.al. [Avritzer *et al.*, 1996] studied the 857 different inputs seen in 355 days operation of an expert system.

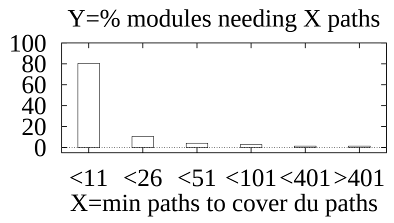

Figure 2: An analysis of hundreds of modules in a software system. 83% of the modules could be fully explored using less than ten tests. From [Bieman and Schultz, 1992]

Program	% coverage			
	Block	Decision	p-use	c-use
TEX	85	72	53	48
AWK	70	59	48	55

Figure 3: Coverage reported by [Horgan and Mathur, 1996, p544]. "Block"= program blocks. "Decision"= program conditionals. "P-use"= pathways between where a variable is assigned and where it is used in a conditional. "C-use"= pathways between where a variable is assigned and where it is used, but not in a conditional.

Massive overlap existed between these input sets. On average, the overlap between two randomly selected inputs was 52.9%. Further, a simple algorithm found that 26 carefully selected inputs covered 99% of the other inputs while 53 carefully selected inputs covered 99.9% of the other inputs.

Harrold et.al. [Harrold *et al.*, 1998] studied how control-flow diagrams grow as program size grows. A worst-case control-flow graph is one where every program statement links to every other statement; i.e. the edges in graph grow with the square of the number of statements. However, for over 4000 Fortran routines and 3147 "C" functions, the control flow graph grows linearly with the number of statements. That is, at least in the studied systems, program pathways form single-parent trees and not complicated tangles.

Horgan and Mathur [Horgan and Mathur, 1996] noted that testing often exhibits a *saturation* effect; i.e. most program paths get exercised early with little further improvement as testing continues. Saturation is consistent with programs containing large portions with simple shapes that are easily reached and other large portions that are so twisted in shape, that they will never be reachable. They report studies with the Unix report-generation language AWK [Aho *et al.*, 1988]) and the word processor TEX [Knuth, 1984]. Both AWK and TEX have been tested extensively for many years by their authors, with the assistance of a vast international user group. Elaborate test suites exist for those systems (e.g. [Knuth, 1984]). Even after elaborate testing, large portions of TEX and AWK were not covered (see Figure 3).

Menzies, Easterbrook, Nuseibeh and Waugh surveyed the internal complexities of a search by compared two *abductive inference strategies*. Abduction is a method of tracking the choices made while studying a model. An abductive inference engine searches for goals while ensuring that all choices remain compatible [Kakas *et al.*, 1998]. When faced with incompatible choices, an abductive device has at least two choices. In *full worlds search*, the abductive device forks one *world of belief* for each possible resolution to the choice. In *stochastic worlds search*, the abductive device selects one resolution at random, then continues on. Stochastic worlds search is usually performed inside a "rest-retry" mechanism. That is, for a limited number of retries, when the stochastic search runs out of new options, all options are retracted and the whole stochastic worlds inference procedure runs again. In a very large case study (over a million runs), Menzies, Easterbrook, Nuseibeh and Waugh found that the average difference in reachable goals between the stochastic worlds search and full worlds search was less than 6% (!!); see Figure 4. That is, stochastic conflict resolution reached as many parts of an argument space as a more rigorous method. This result is consistent with a belief that the theories they studied contained simple search pathways.

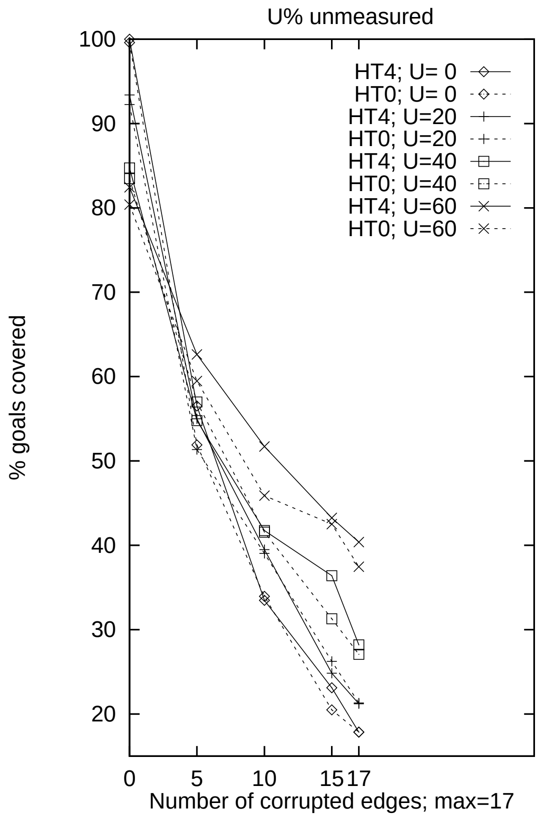

Figure 4: Rigorous standard abductive search (solid line) vs stochastic abductive search (dashed line). Y-axis denotes percentage of goals reached. As errors are introduced into the theory (increasing the x-axis value), this percentage decreases. *U* denotes what percentage of the theory was not mentioned in the input-output sets.

In yet another study comparing stochastic vs full worlds search, Menzies and Micheal [Menzies and Michael, 1999] showed that stochastic worlds search found 98% of the goals found by a full worlds search [Menzies and Michael, 1999] (a result consistent with Menzies, Easterbrook, Nuseibeh and

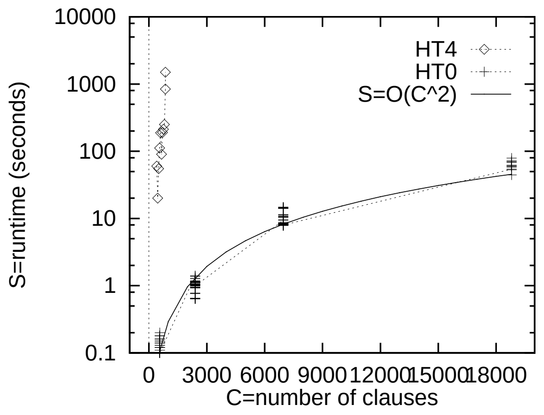

Figure 5: Rigorous standard abductive search (HT4) vs stochastic abductive search (HT0).In the zone where both terminate, HT0 reached 98% of the goals reached by HT4.

```
happy             if tranquillity(hi)
                     or rich and healthy.
healthy           if diet(light).
satiated          if diet(fatty).
tranquillity(hi) if satiated
                    or conscience(clear)
diet(fatty).
diet(light).

% contradiction knowledge;  e.g. diet(fatty)
% and diet(light) are nogood.
nogood(X,Y) :-
   X =.. [F|A1], Y =.. [F|A2], A1 \= A2.
```

Figure 6: A theory.

Waugh). More interesting from a pragmatic perspective, the full worlds search ran in time exponential to model size while the stochastic abductive search can run much faster and scaled up to very large models (see Figure 5). This result can be explained as above. If the search space of a program is basically simple, then the stochastic worlds search used by Menzies & Micheal ran extremely fast since it could quickly sample the space without all the overheads of the more rigorous search.

4 Simulations of Approximate Testing

If these case studies represented the general case, then we should have great confidence in the utility of approximate testing. The average shape of software can be inferred from the odds of reaching any part of the system from random input. If the odds are high, then the pathways to that part must be simple. To infer these odds, Menzies & Cukic [Menzies and Cukic, 2000] assumed that software had been transformed into a possibly cyclic directed graph containing and-nodes and or-nodes (e.g. Figure 6 would be converted to Figure 7).

A simplified description of their analysis is presented here. For reasons of space, that simple description ignores certain details presented in the full description of the model [Men-

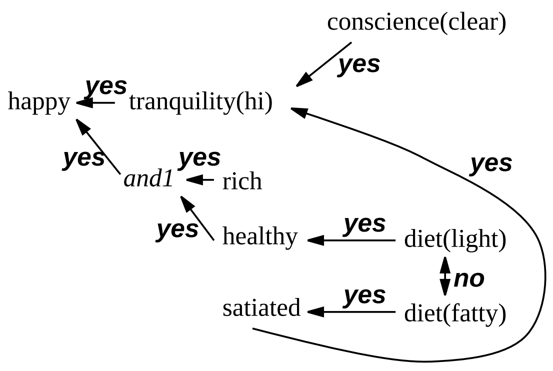

Figure 7: The rules of Figure 6 converted to an and-or graph. All nodes here are or-nodes except *and1*. All parents of an and-node must be believed if we are to believe and and-node. In this graph *no-edges* represent illegal pairs of inferences; i.e. things we can't believe at the same time such as `diet(light)` and `diet(fatty)`. All other edges are *yes-edges* which represent legal inferences.

zies *et al.*, 2000] such as random variables, and testing for loops/contradictions.

To compute the odds of reaching some part of a program graph, we need tools. Our first tool is the standard *sampling-with-replacement* expression:

$$y = \quad 1 - \left((1 - x_j)^N\right) \tag{1}$$

To derive this expression, recall that an event with probability x does not happen after N trials with probability $(1-x)^N$. Hence, at probability y, the event will happen with the probability shown in Equation 1. This equation assumes test independence; i.e. the effects of performing one test do not affect the others.

Our second tool is an average case analysis of the *reachability* of programs. Assume that "in" number of inputs have been presented to a graph containing V nodes. From these inputs, we grow a tree of pathways down to some random node within the graph. The odds of reaching a node straight away from the inputs is $x_0 = \frac{in}{V}$. The probability of reaching an and-node with *andp* parents is the probability of reaching all its parents; i.e. $x_{and} = x_i^{andp}$ where x_i is the probability we computed in the prior step of the simulation (and x_0 being the base case). The probability of reaching an or-node with *orp* parents is the probability of not missing any of its parents; i.e. $x_{or} = 1 - (1 - x_i)^{orp}$ (via Equation 1). If the ratio of and-nodes in a network is $andf$, then the ratio of or-nodes in the same network is $1-andf$. The odds of reaching some random node x_j is the weighted sum of the probabilities of reaching and-nodes or or-nodes; i.e. $x_j = andf * x_{and} + orf * x_{or}$. We can convert x_j to the number of tests N required to be 99% sure of find a fault with probability x_j by rearranging Equation 1 to:

$$N \quad = \quad \frac{log(1-0.99)}{log(1-x_j)} \tag{2}$$

After 150,000 simulations of this model, the number of random inputs required to be 99% sure of reaching a node were usually either surprisingly small or impractically large:

- In 55% of the runs, less than 100 random tests had a 99% chance of reaching any node. This result is consistent with numerous simple i/o pathways.
- In 20% of the runs, the number of random tests required to be 99% sure of reaching any node was between one million and 10^{14}. This result is consistent with large unreachable regions.

In the remaining 25% of cases, systems needed between 10,000 and 1,000,000 random tests to be probed adequately. In these remaining cases, a few approximate tests would be inadequate to probe a system.

The good news from this simulation is that for most systems ($55 + 25 = 75\%$), a small number of tests will yield as much information as an impossibly large number of tests. For these systems:

- There is no point conducting lengthy and expensive testing since a limited testing regime will yield as much information as an elaborate testing procedure.
- Approximate testing is an adequate test regime.

The bad news from this simulation is twofold. Firstly, the Menzies & Cukic model is an average-case analysis of the recommended effort associated with testing. By definition, such an average case analysis says little about extreme cases of high criticality. Hence, our analysis must be used with care if applied to safety-critical software. Secondly, according to this model, approximate testing is inadequate in at least 25% of the space of systems explored by Menzies & Cukic. Hence, for those 25% of systems and for safety-critical systems, we need alternatives to approximate testing.

References

[Aho *et al.*, 1988] A.V. Aho, B.W. Kernigham, and P.J. Wienberger. *The AWK Programming Language*. Addison-Wesley, 1988.

[Avritzer *et al.*, 1996] A. Avritzer, J.P. Ros, and E.J. Weyuker. Reliability of rule-based systems. *IEEE Software*, pages 76–82, September 1996.

[Bahill *et al.*, 1995] A.T. Bahill, K. Bharathan, and R.F. Curlee. How the testing techniques for a decision support systems changed over nine years. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(12):1535–1542, December 1995.

[Betta *et al.*, 1995] G. Betta, M. D'Apuzzo, and A. Pietrosanto. A knowledge-based approach to instrument fault detection and isolation. *IEEE Transactions of Instrumentation and Measurement*, 44(6):1109–1016, December 1995.

[Bieman and Schultz, 1992] J.M. Bieman and J.L. Schultz. An empirical evaluation (and specification) of the all-du-paths testing criterion. *Software Engineering Journal*, 7(1):43–51, 1992.

[Bobrow *et al.*, 1986] D.G. Bobrow, S. Mittal, and M.J. Stefik. Expert systems: Perils and promise. *Communications of the ACM*, 29:880–894, 1986.

[Buchanan *et al.*, 1983] B. Buchanan, D. Barstow, R. Bechtel, J. Bennet, W. Clancey, C. Kulikowski, T.M. Mitchell, and D.A. Waterman. *Building Expert Systems, F. Hayes-Roth and D. Waterman and D. Lenat (eds)*, chapter Constructing an Expert Sytem, pages 127–168. Addison-Wesley, 1983.

[Caraca-Valente *et al.*, 2000] J.P. Caraca-Valente, L. Gonzalez, J.L. Morant, and J. Pozas. Knowledge-based systems validation: When to stop running test cases. *International Journal of Human-Computer Studies*, 2000. To appear.

[Colomb, 1999] R.M. Colomb. Representation of propositional expert systems as partial functions. Artificial Intelligence (to appear), 1999. Available from `http://www.csee.uq.edu.au/~colomb/PartialFunctions.html`.

[Davies, 1994] P. Davies. Planning and expert systems. In *ECAI '94*, 1994.

[Harmon and King, 1983] D. Harmon and D. King. *Expert Systems: Artificial Intelligence in Business*. John Wiley & Sons, 1983.

[Harrold *et al.*, 1998] M.J. Harrold, J.A. Jones, and G. Rothermel. Empirical studies of control dependence graph size for c programs. *Empirical Software Engineering*, 3:203–211, 1998.

[Horgan and Mathur, 1996] J. Horgan and A. Mathur. Software testing and reliability. In M. R. Lyu, editor, *The Handbook of Software Reliability Engineering*, pages 531–565, McGraw-Hill, 1996.

[Kakas *et al.*, 1998] A.C. Kakas, R.A. Kowalski, and F. Toni. The role of abduction in logic programming. In C.J. Hogger D.M. Gabbay and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, pages 235–324. Oxford University Press, 1998.

[Knuth, 1984] D.E. Knuth. A torture test for TEX. Technical Report STAN-CS-84-1027, Department of Computer Science, Stanford University, 1984.

[Menzies and Cukic, 2000] T. Menzies and B. Cukic. When to test less. *IEEE Software*, 17(5):107–112, 2000. Available from `http://tim.menzies.com/pdf/00iesoft.pdf`.

[Menzies and Cukic, 2001] T.J. Menzies and B. Cukic. How many tests are enough? In S.K. Chung, editor, *Handbook of Software Engineering and Knowledge Engineering, Volume II*. World-Scientific, 2001. Available from `http://tim.menzies.com/pdf/00ntests.pdf`.

[Menzies and Michael, 1999] T. Menzies and C.C. Michael. Fewer slices of pie: Optimising mutation testing via abduction. In *SEKE '99, June 17-19, Kaiserslautern, Germany. Available from* `http://tim.menzies.com/pdf/99seke.pdf`, 1999.

[Menzies *et al.*, 2000] Tim Menzies, Bojan Cukic, Harhsinder Singh, and John Powell. Testing nondeterminate systems. In *ISSRE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00issre.pdf`.

[Menzies, 1998] T.J. Menzies. Evaluation issues with critical success metrics. In *Banff KA '98 workshop.*, 1998. Available from `http://tim.menzies.com/pdf/97langevl.pdf`.

[Ramsey and Basili, 1989] C. Loggia Ramsey and V.R. Basili. An evaluation for expert systems for software engineering management. *IEEE Transactions on Software Engineering*, 15:747–759, 1989.

[Yu *et al.*, 1979] V.L. Yu, L.M. Fagan, S.M. Wraith, W.J. Clancey, A.C. Scott, J.F. Hanigan, R.L. Blum, B.G. Buchanan, and S.N. Cohen. Antimicrobial Selection by a Computer: a Blinded Evaluation by Infectious Disease Experts. *Journal of American Medical Association*, 242:1279–1282, 1979.