Intelligent Testing can be Very Lazy

Tim Menzies[1], **Bojan Cukic**[2]

[1]NASA/WVU Software Research Lab, 100 University Drive, Fairmont, WV USA
[2]Department of Computing Science and Electrical Engineering, West Virginia University, Morgantown, WV USA
<tim@menzies.com,cukic@csee.wvu.edu>

Abstract

Testing is a search process and a test suite is complete when the search has examined all the corners of the program. Standard models of test suite sizes are gross over-estimates since they are unaware of the nature of that search space. For example, only a small part of the possible search space is ever exercised in practice. Further, a repeated result is that a few random searches often yields as much information as more thorough search strategies. Hence, only a few tests are needed to sample the range of behaviours of a program[1].

Introduction

Due to recent cutbacks, ISE Ltd. has to fire one of its test engineers. Who would you pick? Lazy Susan never tests a program rigorously. Instead, she just has a quick glance at some of the possible consequences of known inputs. Lazy Susan only ever proposes small test suites based just on small variants to the inputs. Lazy Susan never works overtime and her tests are always finished on the due data. On the other hand, Eager Earnest is more thorough and reflects on how all the different KB pathways can interact and interfere with each other. Earnest often proposes larger test suites based on the inputs, and all the interaction points downstream of the inputs. Earnest always works overtime and never finishes his testing by the due date.

Who gets sacked? Earnest thinks it should be Susan, saying that even a cursory study of the mathematics shows that testing is fundamentally a slow process. For example, a simple attribute model would declare that a system containing N variables with S assignments (on average) requires S^N tests. Further, a widely-used statistical model declares that over 4,500 tests are required to find moderately infrequent bugs; see Figure 1.

Not surprisingly, Susan disagrees with Earnest and thinks he should be the one to go. She defends her test stratgies, noting that most of the expert systems literature proposes evaluations based on very few tests[2]; see Table 1. For example, Menzies needed a mere 40 tests to check an expert system that controlled a complex chemical plant (125 kilometers of highly inter-connected piping) (Menzies 1998). In that design, the expert system and the human operators took turns to run the plant. At the end of a statistically significant number of trials, the mean performance are comparable using a t-test[3]. Further, she rejects Earnest's S^N model as obviously ridiculous as follows:

> In sample of fielded expert systems, knowledge bases contained between 55 and 510 (Preece & Shinghal 1992). Literals offer two assignments for each proposition: true or false (i.e. $S = 2$ and N is half the number of literals. Assuming (i) it takes one minute to consider each test result (which is a gross under-estimate) and (ii) that the effective working year is 225 six hour days, then a test of those sampled systems would take between 29 years and 10^{70} years: a time longer than the age of this universe.

Text	# of tests
(Harmon & King 1983)	4..5
(Buchanan *et al.* 1983)	≈ 6
(Bobrow, Mittal, & Stefik 1986)	5..10
(Davies 1994)	8..10
(Yu *et al.* 1979)	10
(Caraca-Valente *et al.* 1999)	< 13
(Menzies 1998)	40
(Ramsey & Basili 1989)	50
(Betta, D'Apuzzo, & Pietrosanto 1995)	200

Table 1: Number of tests proposed by different authors. Extended from a survey by (Caraca-Valente *et al.* 1999).

[1]A submission to the First International Workshop on Intelligent Software Engineering, Orlando, Florida, July 1999. WP:b/99/ise/talkless(April 30, 1999).

[2]Exception: (Bahill, Bharathan, & Curlee 1995) propose at least one test for every five rules and add that "having more test cases than rules would be best".

[3]Let m and n be the number of trials of expert system and the human experts respectively. Each trial generates a performance score (time till unusual operations): $X_1 \ldots X_m$ with mean μ_x for the humans; and performance scores $Y_1 \ldots Y_n$ with mean μ_y for the expert system. We need to find a Z value as follows: $Z = \frac{\mu_x - \mu_y}{\sqrt{\frac{S_x^2}{m} + \frac{S_y^2}{n}}}$ where $S_x^2 = \frac{\Sigma(x_i - \mu_x)^2}{m-1}$ and $S_y^2 = \frac{\Sigma(y_i - \mu_y)^2}{n-1}$, Let a be the degrees of freedom. If $n = m = 20$, the $a = n + m - 2 = 38$. We reject the hypothesis that expert system is worse than the human (i.e. $\mu_x < \mu_y$) with 95% confidence if Z is less than ($-t_{38,0.95} = -1.645$).

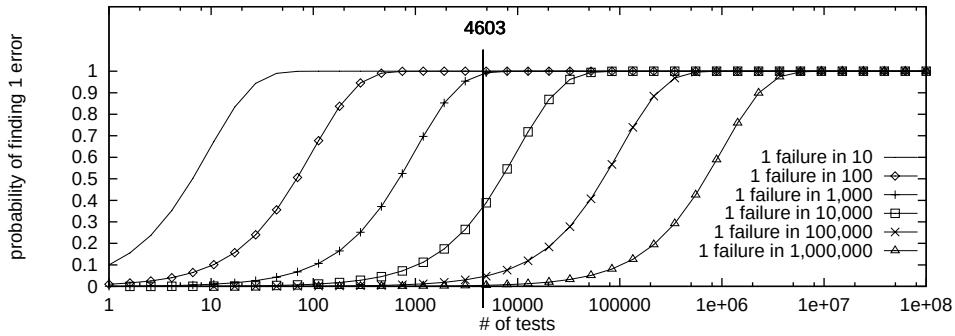

Figure 1: Chance of finding an error = $1 - (1 - failure\ rate)^{tests}$. Theoretically, 4603 tests are required to achieve a 99% chance of detecting moderately infrequent bugs; i.e. those which occur at a frequency of 1 in a thousand cases. (Hamlet & Taylor 1990).

The research division of ISE Inc. has been hired to resolve this dispute. Their core insight is that testing samples a search space. Some parts of the space contain desired goals and some parts contain undesirable behaviour. Regardless of how users interpret different parts of the space, the testing-as-search process is the same: a test suite is complete when it has looked in all the corners of that search space.

This notion of testing=search covers numerous testing schemes:

- Oracle testing uses some external source of information to specify distributions for the inputs and outputs
- Model-checking for temporal properties (Clarke, Emerson, & Sistla 1986) can be reduced to search as follows. First, the model is expressed as the dependency graph between literals. Secondly, the constraints are grounded and negated. The generation of counter-examples by model checkers then becomes a search for pathways from inputs to undesirable outputs.
- A test suite that satisfies the "all-uses" data-flow coverage criteria can generate proof trees for all portions of the search space that connect some literal from where it is assigned to wherever it is used (Frankl & Weiss 1993).
- Partition testing (Hamlet & Taylor 1990) is the generation of inputs via a reflection over the search space looking for key literals that fork the program's conclusions into N different partitions.
- The multiple-worlds reasoners (described below) divide the accessible areas of a search space into their consistent subsets.

Based on this insight of testing=search, ISE Research made the following conclusion. Table 1 proposes far fewer tests than Figure 1 since Figure 1 has limited insight into the devices being testing. Figure 1's estimates are grossly over-inflated since it is unaware of certain regularities within the software it is testing. ISE research bases this conclusion on a literature review which shows:

- While a static analysis of a program suggests S^N states, in practice only a small percentage of those are exercised by the systems inputs.
- Further, a review of inference engines shows that a few random stabs into the search space often yields as much information as more thorough search strategies.

The literature review is presented below. Note that this article is framed in terms of knowledge engineering. However, there is nothing stopping these results from being applied to software engineering (Menzies & Cukic 1999).

Studies with Data Sets

The sections discusses two studies with data sets showing that the attribute space size of the data presented to an expert system is far less than the potential upper bound of the attribute space size of the KB (S^N). This is consistent with the used portions of search spaces being non-complex. Test suites grown beyond some small size would yield no new information since the smaller test suite would have already explored the interesting parts of the KB's search space.

Avritzer *et.al.* studied the inputs given to an expert system. Each row of a state matrix stored one unique case presented to the expert system (and each literal in the inputs were generate one column in the matrix). After examining 355 days of input data, they could only find 857 different inputs. There was massive overlap between those 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 (Avritzer, Ros, & Weyuker 1996). If most naturally occuring test suites have such an overlap, then only a small number of tests will be required.

Avritzer *et.al.* looked at the inputs to a system. Colomb compared the inputs presented to an expert system with its internal structure. Recalling the introduction, the internals of a KB can very large: S states per N variables implies S^N combinations. Each such combination could be represented as one row in a state matrix. Colomb argues that the estimate of S^N is a gross over-inflation of effective KB size. He argues that KBs record the very small *regions of experience* of human experts. For example, one medical expert system studied by Colomb had $S^N = 10^{14}$. However, after one year's operation, the inputs to that expert sys-

```
Isamp(theory) {
for i := 1 to MAX-TRIES {
  set all variables to unassigned;
  loop {
       if all variables are valued
            return(current assignment);
       v := random unvalued variable;
       assign v a randomly chosen value;
       unit_propagate();
       if contradiction exit loop;
  }
 } return failure
}
```

Figure 2: The ISAMP randmoised-search theorem prover. (Crawford & Baker 1994).

tem could be represented in a state matrix with only 4000 rows. That is, the region of experience exercised after one year within the expert system was only a tiny fraction of S^N ($4000 \ll 10^{14}$) (Colomb 1998). If most systems have this feature, then only a few tests will be required to exercise a KB's region of expertise.

Studies with Inference Engines

This section reviews two studies suggesting the extra effort of the thorough engineer would be wasted since the larger thorough test suite would yield little more information that the smaller lazy test suite.

Recall the behaviour of our two test engineers:

- Lazy Susan explored a couple of randomly chosen paths while Earnest considered all inputs and their downstream interactions.
- Earnest is acting like an ATMS device (DeKleer 1986). The ATMS takes the justification for every conclusions and weaves it into an assumption network. At anytime, the ATMS can report the key assumptions that drive the KB into very different conclusions. Such a search, it was thought, was a useful way to explore all the competing options within a search space.
- Lazy Susan is acting like a locally-guided best-first search across the KB. When a contradiction is detected, Susan (or the search engine) could look at the contradiction for hints on how to resolve that problem. Susan (or the search engine) could then instantiated those hints and search on. Note that, as a results, the earnest ATMS would find many options and the lazy searcher would only ever find one.

Williams and Nayak compared the results of their locally-guides search to the ATMS and found that this locally-guide contradiction resolution mechanism was comparable to the very best ATMS implementations. That is, the information gained from an exploration of all options was nearly equivalent to the information gained from the exploration of a single option (Williams & Nayak 1996).

In other work, Crawford and Baker compared TABLEAU, a depth-first search backtracking algorithm, to ISAMP, a randomised-search theorem prover (see Figure 2). ISAMP randomly assigns a value to one variable, then infers some consequences using unit propagation. Unit propagation is not a very thorough inference procedure: it only infers those conclusions which can be found using a special linear-time case of resolution; i.e.

$$(x) \wedge (\neg x \; or \; y_1 \; ... \; y_n) \vdash (y_1 \; or ... \; y_n)$$

$$(\neg x) \wedge (x \; or \; y_1 \; or ... \; y_n) \vdash (y_1 \; or ... \; y_n)$$

After unit propagation, if a contradiction was detected, ISAMP re-assigns all the variables and tries again (giving up after `MAX-TRIES` number of times). Otherwise, ISAMP continues looping till all variables are assigned. Lazy Susan approves of ISAMP while Earnest likes TABLEAU since it explores more options more systematically.

	TABLEAU: full search		ISAMP: partial, random search		
	% Success	Time (sec)	% Success	Time (sec)	Tries
A	90	255.4	100	10	7
B	100	104.8	100	13	15
C	70	79.2	100	11	13
D	100	90.6	100	21	45
E	80	66.3	100	19	52
F	100	81.7	100	68	252

Table 2: Average performance of TABLEAU vs ISAMP on 6 scheduling problems (A..F) with different levels of constraints and bottlenecks. From (Crawford & Baker 1994).

Table 2 shows the relative performance of the two algorithms on a suite of scheduling problems based on real-world parameters. Surprisingly, ISAMP took *less* time than TABLEAU to reach *more* scheduling solutions using, usually, just a small number of `TRIES`. That is, a couple of random explorations of the easy parts of a search space yielded the best results. Crawford and Baker offer a speculation why ISAMP was so successful: their systems contained mostly "dependent" variables which are set by a small number of "control" variables (Crawford & Baker 1994). If most systems have this feature, then large test suites are not required since a few key tests are sufficient to set the control variables.

Experiments with HTx

The results in the previous section support the argument that search spaces are less complex than Earnest thinks. Hence, the test suites required to sample that search space may be very small. However, the sample size of the above studies is very small. This section addresses the sample size issue. Based on the HTx *abductive model of testing* (Menzies 1995; Menzies & Compton 1997), a suite of *mutators* can generate any number of sample testing problems[4]. Three

[4]Informally, abduction is inference to the best explanation. More precisely, abduction make whatever assumptions A which are required to reach output goals Out across a theory $T \cup A \vdash Out$ without causing contradiction $T \cup A \nvdash \perp$. Each consistent set of assumptions represents an explanation. If more than one explanation is found, some assessment criteria is applied to select the preferred explanation(s).

HTx algorithms are HT4 (Menzies 1995), HT4-dumb (Menzies & Waugh 1998), and HT0 (Menzies 1999). HTx algorithms were designed as automatic hypothesis testers for under-specified theories and are a generalization and optimization of QMOD, a validation tool for neuroendocrinological theories (Feldman, Compton, & Smythe 1989). HTx and QMOD assumes that the definitive test for a theory is that it can reproduce (or $cover$) known behaviour of the entity being modeled. Theory T_1 is a better theory than theory T_2 iff

$$cover(T_1) \gg cover(T_2)$$

Below, we will show experiments where HTx is run millions of times over tens of thousands of models. The results endorse the widespread existence of simple search spaces. A small number of random searches (by HT0) covered nearly as much output as extensive searches by HT4. Further, the sample size is much larger than that offered by Williams, Nayak, Crawford, and Baker.

An Informal Model of Testing

This section gives a quick overview of the intuitions of HTx. The next section details these intuitions.

At runtime, an inference engine explores the search space of a program. A given test applies some inputs (In) to the program to reach some outputs (Out). The inference engine may be indeterminate or chosen randomly (e.g. by ISAMP) and so the pathway taken from inputs to outputs may vary. Hence, we call the intermediaries *assumptions* (A). That is:

$$< Out, A >= f(In, KB)$$

where f is the inference engine and KB are the literals $\{a..z\}$ which we can divide as follows:

$$\overbrace{a, b, c, ..}^{In} \underbrace{..., l, m, n, ...,}_{A} \overbrace{...x, y, z}^{Out}$$

That is, from the inputs $a, b, c, ...$, we can reach the outputs $...x, y, z$ via the assumptions $...l, m, n, ...$ Some of $a..z$ are positive goals that we are trying to achieve while some are negative goals reflecting situations we are trying to avoid. In the general case, only a subset of Out will be reachable using some subset of the In, A since:

- The search space may not approve of connections between all of In and Out.
- Some of the assumptions may be contradictory. Multiple worlds of beliefs may be generated when contradictory assumptions are sorted into their maximal consistent subsets.
- In the case of randomized search, only parts of the search space may be explored. For example, we could run ISAMP for a finite number of `TRIES` and return the `TRY` which the maximum number of assignments to Out.

Our testing intuition is that a test case beams a searchlight from In across A to Out. Sometimes, the light reveals something interesting. We can stop testing when our searchlight stops finding anything new. What will be shown below is that a few quick flashes show as much as lots of poking around in corners with the flashlight.

HTx: An More Formal Model of Testing

This section describes the details of HTx. The next section describes an example.

We represent a theory as a directed-cyclic graph $T = G$ containing vertices $\{V_i, V_j, ...\}$ with roots $roots(G)$ and leaves $leaves(G)$. A vertex is one of two types:

- *Ands* can be believed if *all* their parents are believed.
- *Ors* can be believed if any their parents are believed or it has been labeled an In vertex (see below). Each or-node contradicts 0 or more other or-nodes, denoted $no(V_i) = \{V_j, V_k, ...\}$. The average size of the no sets is called $contraints(G)$. For propositional systems, $contraints(G) = 1$; e.g. $no(a) = \{\neg a\}$.

In and Out are sets of vertices from $ors(G)$. A proof $P \subseteq G$ is a tree containing the vertices $uses(P_x) = \{V_i, V_j, ...\}$.The proof tree has:

- Exactly one leaf which is an output; i.e.

$$|leaves(uses(P_x))| = 1 \wedge$$

$$V_i \in leaves(uses(P_x)) \wedge V_i \in Out$$

- 1 or more roots $roots(uses(P_x)) \subseteq In$.

No two proof vertices can contradict each other; i.e.

$$< V_i, V_j > \in uses(P_x) \wedge V_j \notin no(V_i)$$

A proof's assumptions are the vertices that are not inputs or outputs; i.e.

$$assumes(P_x) = uses(P_x) - roots(uses(P_x)) - leaves(uses(P_x))$$

A test suite is N pairs $\{< In_1, Out_1 >, ... < In_n, Out_n >$. We assumed that each In_i, Out_i contains only or-vertices. Each output Out_i can generated 0 or more proofs $\{P_x, P_y, ...\}$. A world is a maximal consistent subsets of the proofs (maximal w.r.t. size and consistent w.r.t. the no sets) denoted $proofs(W_i) = \{P_x, P_y, ...\}$. Worlds to proofs is many-to-many. The cover of a world is how many outputs it contains; i.e.

$$cover(W_i) = \frac{\left| \bigcup_{V_y} \{P_x \in proofs(W_i) \wedge V_y \in leaves(P_x)\} \right|}{|Out|}$$

We make no other comment on the nature of Out_i. It may be some undesirable state or some desired goal. In either case, the aim of our testing is to find the worlds that cover the largest percentage of Out.

An Example

Figure 3 (left) shows a hypothetical economics theory written in the QCM language (Menzies & Compton 1997). All theory variables have three mutually exclusive states: *up*, *down* or *steady*; i.e. $no(V_{a=up}) = \{V_{a=down}, V_{a=steady}\}$ and $constraints(G) = 2$. These values model the sign of the first derivative of these variables (i.e. the rate of change in each value). In QCM, $x \stackrel{++}{\rightarrow} y$ denotes that *y* being *up*

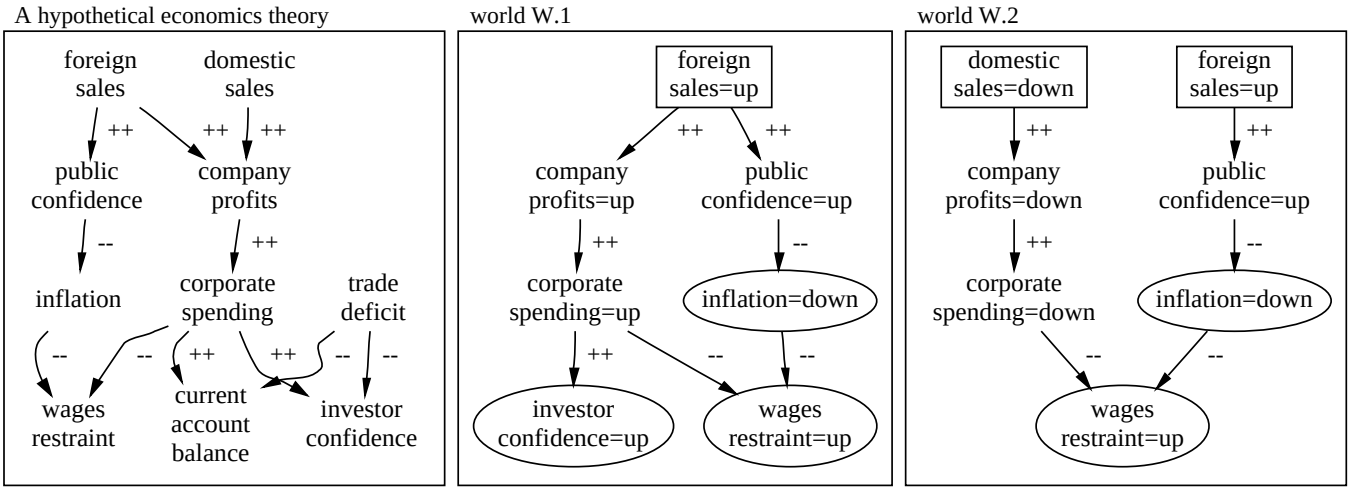

Figure 3: Two worlds for outputs (ellipses) from inputs (squares). Assumptions are world vertices that are not inputs or outputs. Note that contradictory assumptions are managed in separate worlds.

P_1	*foreignSales=up, companyProfits=up†, corporateSpending=up†, investorConfidence=up*
P_2	*domesticSales=down, companyProfits=down†, corporateSpending=down†, wageRestraint=up*
P_3	*domesticSales=down, companyProfits=down†, inflation=down*
P_4	*domesticSales=down, companyProfits=down†, inflation=down, wagesRestraint=up*
P_5	*foreignSales=up, publicConfidence=up, inflation=down*
P_6	*foreignSales=up, publicConfidence=up, inflation=down, wageRestraint=up*

Table 3: Proofs connecting inputs to outputs. *X*†, and *X*†denote the assumptions and the controversial assumptions respectively.

or *down* could be explained by *x* being *up* or *down* respectively. Also, $x \stackrel{--}{\rightarrow} y$ denotes that *y* being *up* or *down* could be explained by *x* being *down* or *up* respectively. Some links are conditional on other factors; for example *if transport then education* $\stackrel{++}{\rightarrow}$ *literacy*. To process such a link, QCM adds a conjunction between *education* and *literacy*. The preconditions of that connection are now *education* and *transport*. QCM also adds conjunctions to offer explanations of *steadies*: the conjunction to competing upstream influences can explain a *steady*.

Consider the case where the inputs In is *(foreignSales=up, domesticSales=down)* and the goals Out are *(investorConfidence=up, inflation=down, wageRestraint=up)*. There are six proofs that connect In to Out (see Table 3). These proofs comprise only or-nodes. These proofs contain assumptions (variable assignments not found in $In \cup Out$). Some of these proofs make contradictory assumptions A; e.g. *corporateSpending=up* in P_1 and *corporateSpending=down* in P_2. That is, we cannot believe in P_1 and P_2 at the same time. If we sort these proofs into the subsets which we can believe at one time, we get worlds W_1 (Figure 3, middle) and W_2 (Figure 3, right). W_1 is a maximal consistent subset of pathways that can be believed at the same time; i.e. $\{P_1, P_5, P_6\}$. W_2 is another maximal consistent subset: $\{P_2, P_3, P_4, P_6\}$. The cover of W_1 is 100% while the cover of W_2 is 67%.

Recall that HTx scores a theory by the maximum cover of the worlds it generates. Hence, our economics theory gets full marks: 100%. HT4 (and QMOD before it) found errors in a published theory of neuroendocrinological (Smythe 1989) when its maximum cover was found to be 42%. That is, after making every assumption possible to explain as many of the goals as possible, only 42% of certain published observations of human glucose regulation could be explained. Interestingly, these faults were found using the data published to support those theories. This suggests that HT4-style abductive validation is practical for assessing existing scientific publications. HTx is practical for many other real-world theories. For example, one implementation of HT4, was fast enough to process at least one published sample of real-world expert systems (Menzies 1996).

HT4, HT4-dumb, HTx

The difference between the HTx algorithms is how they searched for their worlds:

- HT4 found all proofs for all outputs, then generated all the worlds from those proofs. Next, the worlds(s) with largest cover was then returned.
- HT4-dumb was an a crippled version of HT4 that returned any world at all, chosen at random. It was meant to be a straw-man system but its results were so promising (see below) that it lead to the development of HT0.
- HT0 is a randomised each engine that `MAX-TRIES` times, finds 1 proof for each member of Out_i. Out is explored in a random order and when the proof is being

1.	Added influences between variables.
2.	Corrupted the influence between two variables; e.g. proportionality ++ was flipped to inverse proportionality - - or visa versa.
3.	Experimented with different meanings time within the system. In the *explicit node* (XNODE) interpretation of time, all $\frac{dX}{dT}$ variables implied a link $(x\ at\ Time^i) \stackrel{++}{\rightarrow} (x\ at\ Time^{i+1})$. In the *implicit edge* IEDGE interpretation of time, all edges $x \stackrel{\alpha}{\rightarrow} y$ also implied a link $(x\ at\ Time^i) \stackrel{\alpha}{\rightarrow} (y\ at\ Time^{i+1})$; $(\alpha \in \{++, --\})$. XNODE and IEDGE are contrasted in Figure 4. Why these two interpretations of time? These were randomly selected from a much larger set of time interpretations studied by (Waugh, Menzies, & Goss 1997).
4.	Forced the algorithm to generate different numbers of worlds. Experiments showed that the maximum number of worlds were generated when between a fifth to three-fifths of the variables in the theory were unmeasured; i.e. U=20..60 where $U = 100 - \frac{(\lvert In\rvert + \lvert Out\rvert) * 100}{\lvert V\rvert}$ and $\lvert V\rvert$ is the number of variables in the theory. The mutators built In and Out sets with different U settings. Values for In and Out were collected using a mathematical simulation of the fisheries.

Table 4: Problem mutators used in the HT4 vs HT4-dumb study

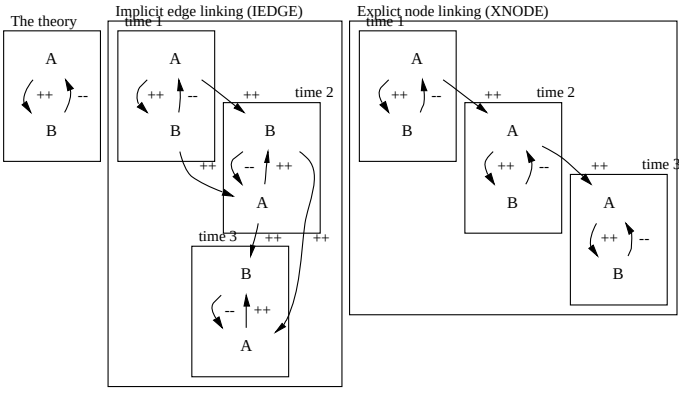

Figure 4: An expansion of the left-hand-side QCM theory into IEDGE and XNODE. Note that for the XNODE expansion, the $\frac{dx}{dt}$ variable was A.

generated, if more than one option is found, one is picked at random. If the proof for Out_j is consistent with the proof found previously for $Out_i, (i < j)$, it is kept. Otherwise, HT0 moves on to $Out_k (j < k)$ and declares Out_j unsolvable. After each `TRY`, HT0 compares the "best" world found in previous `TRIES` to the world found in the try and discards the one with the lowest cover.

Susan approves of HT0 since, like her, it uses a lazy method to study a program. Earnest likes HT4 since it explores more of the program. For example, says Earnest, if Lazy Susan had stumbled randomly on W_2 of Figure 3, and did not look any further to find W_1, then she would have inappropriately declared that the economics model could only explain 67% of the outputs. Susan replies that, on average, the cost of such extra searching is not justified by its benefits. The following studies comparing HTx algorithms support Susan's case.

HT4 vs HT4-dumb

Menzies and Waugh compared HT4 and HT4-dumb using tens of thousands of theories (Menzies & Waugh 1998). Starting with a seed theory (fish growing in a fishery, see Figure 5), automatic mutators were built to generate a wide range of problems (see Table 4). When these problems were run with HT4 and HT4-dumb, the maximum difference in the cover was 5.6%; i.e. very similar, see Figure 6. That is, in a large number (1,512,000) of world generation experiments, (1) many different searches contain the same goals, (2) there was little observed utility in using more than one world.

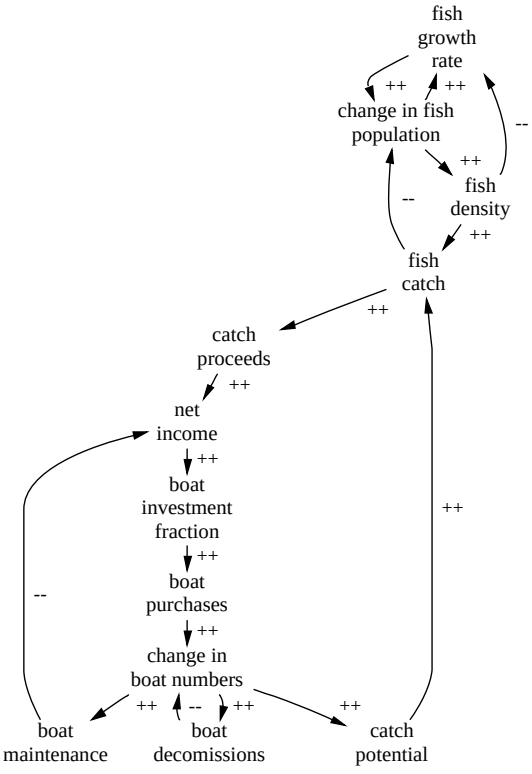

Figure 5: The QCM fisheries model contains two $\frac{dX}{dT}$ variables: *change in fish population* and *change in boat numbers.*

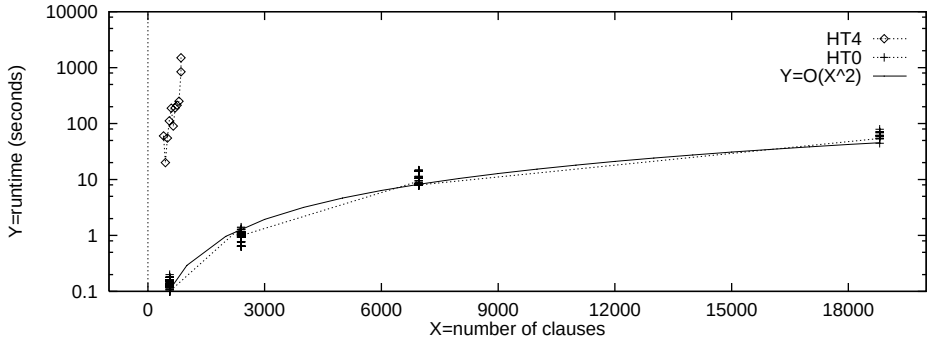

Figure 7: Runtimes HT4 (top left) vs HT0 (bottom). HT0 was observed to be $O(N^2)$ ($R^2 = 0.98$).

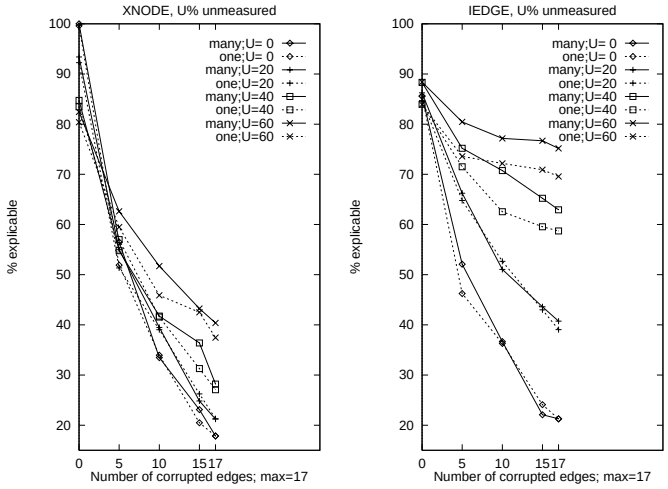

Figure 6: Comparing coverage of Out seen in 1,512,000 runs of the HT4 (solid line) or HT4-dumb (dashed line) algorithms with different percentages of unmeasured U variables. X-axis refers to how many edges of Figure 5 were corrupted (`++`, `--` flipped to `--`, `++` respectively.)

HT4 vs HT0

HT4-dumb was implemented as a back-end to HT4. HT4 would propose lots of worlds and HT4-dumb would pick one at random. That is, HT4-dumb was at least as slow as HT4. However, it worked so astonishingly well, that HT0 was developed. Real world and artificially generated theories were used to test HT0. A real-world theory of neuroendocrinology with 558 clauses containing 91 variables with 3 values each (273 literals) was copied X times. Next, Y% of the variables in in one copy were connected at random to variables in other copies. In this way, theories between 30 and 20000 Prolog clauses were built using Y=40, average $\frac{sub-goals}{clause} \approx 1.7$, $\frac{clauses}{literals} = 1..5.5$. When executed with `MAX-TRIES=50` the O(N^2) curve of Figure 7 was generated. HT4 did not terminate for theories with more than 1000 clauses. Hence, we can only compare the cover of HT0 to HT4 for part of this experiment. In the comparable region, HT0 found 98% of the outputs found by HT4.

In other experiments, the maximum cover found for different values of `MAX-TRIES` was explored. Surprisingly, the maximum cover found for `MAX-TRIES=50` was found with `MAX-TRIES` as low as 6.

In summary, a small number of quick random searches (HT0) found nearly as much of the interesting portions of the KB as a careful, slower, larger number of searchers (HT4).

Conclusion

On average, exploring a theory is not as complex as one might think. Testing is a search process and a test suite is complete when the search has examined all the corners of the program. Often a few lazy explorations of a search space yields as much information as more thorough searches.

Philosophically, this must be true. Our impressions are that human resources are limited and most explorations of theories are cost-bounded. Hence, many theories are not explored rigourously. Yet, to a useful degree, this limited reasoning about our ideas works. This can only be true if our theories yield their key insights early to our limited tests. Otherwise, we doubt that the human intellectual process could have achieved so much.

Earnest strongly disagrees with the last paragraph, saying that such an argument represents the height of human arrogance; i.e. once again humanity is claiming to have some special authority over the random universe around us. Lazy Susan does not have the heart to disagree. She'd just been promoted while Earnest had been sacked. Her view is that this arrogance seems justified, at least based on the above evidence.

She had lunch with Earnest on his last day. As he cried into his soup, he warned that the above analysis is overly optimistic. The results of this article refer to the average case behaviour of a test rig. In safety-critical situations, such an average case analysis is inappropriate due to the disastrous implications of the non-average case. Secondly, these results only refer to the number of tests that can detect anomalous behaviour. Once an error is found, it must be localized and removed. Strategies for fault localization and repair are explored elsewhere in the literature (Shapiro 1983; Hamscher, Console, & DeKleer 1992).

Susan paid for that last lunch using her pay rise: it seemed the decent thing to do.

Acknowledgements

This work was partially supported by NASA through cooperative agreement #NCC 2-979.

References

Avritzer, A.; Ros, J.; and Weyuker, E. 1996. Reliability of rule-based systems. *IEEE Software* 76–82.

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

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

Bobrow, D.; Mittal, S.; and Stefik, M. 1986. Expert systems: Perils and promise. *Communications of the ACM* 29:880–894.

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

Caraca-Valente, J.; Gonzalez, L.; Morant, J.; and Pozas, J. 1999. Knowledge-based systems validation: When to stop running test cases. *International Journal of Human-Computer Studies*. To appear.

Clarke, E.; Emerson, E.; and Sistla, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244–263.

Colomb, R. 1998. Representation of propositional expert systems as partial functions. Submitted to AIJ, Feb 10, 1998. Available from `http://www.csee.uq.edu.au/~colomb/PartialFunctions.html`.

Crawford, J., and Baker, A. 1994. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*.

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

DeKleer, J. 1986. An Assumption-Based TMS. *Artificial Intelligence* 28:163–196.

Feldman, B.; Compton, P.; and Smythe, G. 1989. Hypothesis Testing: an Appropriate Task for Knowledge-Based Systems. In *4th AAAI-Sponsored Knowledge Acquisition for Knowledge-based Systems Workshop Banff, Canada*.

Frankl, P., and Weiss, S. 1993. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Transactions on Software Engineering* 19(8):774–787.

Hamlet, D., and Taylor, R. 1990. Partition testing does not inspire confidence. *IEEE Transactions on Software Engineering* 16(12):1402–1411.

Hamscher, W.; Console, L.; and DeKleer, J. 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann.

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

Menzies, T., and Compton, P. 1997. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine* 10:145–175. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96aim.ps.gz`.

Menzies, T., and Cukic, B. 1999. When you don't need to re-test the system. In *Submitted to ICRE-99*. In preperation.

Menzies, T., and Waugh, S. 1998. On the practicality of viewpoint-based requirements engineering. In *Proceedings, Pacific Rim Conference on Artificial Intelligence, Singapore*. Springer-Verlag.

Menzies, T. 1995. *Principles for Generalised Testing of Knowledge Bases*. Ph.D. Dissertation, University of New South Wales. Avaliable from `http://www.cse.unsw.edu.au/~timm/pub/docs/95thesis.ps.gz`.

Menzies, T. 1996. On the practicality of abductive validation. In *ECAI '96*. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96abvalid.ps.gz`.

Menzies, T. 1998. Evaluation issues with critical success metrics. In *Banff KA '98 workshop*. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/97evalcsm`.

Menzies, T. 1999. Simpler, faster abductive validation. *AAAI-99 (submitted)*.

Preece, A., and Shinghal, R. 1992. Verifying knowledge bases by anomaly detection: An experience report. In *ECAI '92*.

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

Shapiro, E. Y. 1983. *Algorithmic program debugging*. Cambridge, Massachusetts: MIT Press.

Smythe, G. 1989. Brain-hypothalmus, Pituitary and the Endocrine Pancreas. *The Endocrine Pancreas*.

Waugh, S.; Menzies, T.; and Goss, S. 1997. Evaluating a qualitative reasoner. In Sattar, A., ed., *Advanced Topics in Artificial Intelligence: 10th Australian Joint Conference on AI*. Springer-Verlag.

Williams, B., and Nayak, P. 1996. A model-based approach to reactive self-configuring systems. In *Proceedings, AAAI '96*, 971–978.

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