Adequacy of Limited Testing for Knowledge Based Systems

Tim Menzies[‡], Bojan Cukic[†]

[‡]NASA/WVU Software Research Lab, 100 University Drive,Fairmont, WV USA

[†]Dept. Computer Science and Electrical Engineering, West Virginia Uni., Morgantown, WV USA

<tim@menzies.com, cukic@csee.wvu.edu>

April 10, 2000

Abstract

Knowledge-based engineering and computational intelligence are expected to become core technologies in the design and manufacturing for the next generation of space exploration missions. The literature is contradictory on how we are to assess such systems. Studies indicate significant disagreement regarding the amount of testing needed for system assessment. The sizes of standard black-box test suites are impractically large since the black-box approach neglects the internal structure of knowledge-based systems. On the contrary, practical results repeatedly indicate that only a few tests are needed to sample the range of behaviors of a knowledge-based program.

In this paper, we model testing as a search process over the internal state space of the knowledge-based system. When comparing different test suites, the test suite that examines larger portion of the state space is considered more complete. Our goal is to investigate the trade-off between the completeness criterion and the size of test suites. The results of testing experiment on tens of thousands of mutants of real-world knowledge based systems indicate that a very limited gain in completeness can be achieved through prolonged testing. The use of simple (or random) search strategies for testing appears to be as powerful as testing by more thorough search algorithms.

Keywords: Software Testing, Knowledge-Based Systems, Software Reliability, Abductive Testing.

1 Introduction

The knowledge based (KB) software paradigm is becoming increasingly popular for building scientific and industrial strength applications. For example, real-time expert systems, embedded in process control applications, must plan and execute control sequences in response to external events within time constraints. The requirement that a control system acts independently of human inputs is considered by NASA as being central for a new generation of intelligent automated space explorers [17]. Applications of knowledge based systems in several engineering disciplines have been widely

reported. One of the basic reasons behind increased use of knowledge based systems is the productivity advantage offered by computer-based decision making to a variety of problem solving tasks.

How are we to assess such systems? Much has been written on syntactic verification techniques for checking if the system was built right. Such verification techniques can detect (e.g.) circularities, contradictions, tautologies, etc. in rule-based systems [38]. However, there is little consensus over methods for semantic validation [47]; i.e., answering the question whether the right system was built? There is no widely adopted strategy for verification and validation of knowledge based systems. Developers test their systems with available test cases. These tests are highly informal, and their outcomes provide nothing more than a "warm and fuzzy feeling" about the quality of the system under test. Several studies have been conducted with the goal of defining the methodology for the reliability assessment of real-time expert systems. Bastani and Chen [4], and later Chen and Tsao [13, 12] noted that the reliability assessment of artificial intelligence (AI) programs must consider not only possible faults in the program text, but also faults due to intrinsic characteristics of AI programs. These intrinsic characteristics include the fuzzy correctness criterion, and the faults inherent in the reasoning algorithms utilized by the expert system. Reliability models must be adapted to reflect these characteristics.

The same intrinsic characteristics are the reason why traditional software testing techniques are not adequate for the assessment of knowledge based systems. Most literature on testing knowledge based systems focuses on checking the completeness, consistency and redundancy of the rule base [37]. Arvitzer, Ros and Veyuker [2] address the problem of how to select test cases to thoroughly test the rule base. In these paper, we are interested in answering the following questions:

- What is an adequate number of tests for a given knowledge based system, and
- When to stop testing.

Informally, we consider testing to be a search process and a test suite is complete when the search has examined all the interesting corners of the program space. It has been noted that, typically, only a small part of the possible search space is ever exercised in practice [2]. The assumption is that this part of the search space is interesting for testers, since most of the program runs will execute in it.

Lets assume, for a moment, that the interesting parts of search spaces in knowledge-based systems are not small. Even a cursory study of the mathematics shows then that testing, fundamentally, must be 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. In sample of fielded expert systems, knowledge bases contained between 55 and 510 literals [39]. 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. If, instead of exhaustive testing, a statistical model of testing is used for assessment, as

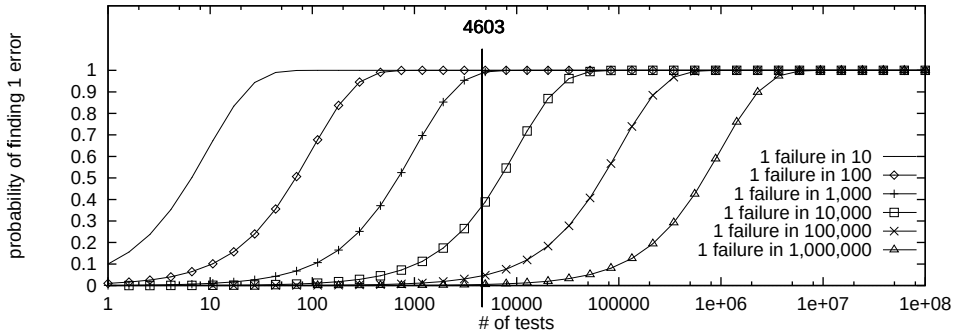

Figure 1: Chance of finding an error = $1 - (1 - failureRate)^{tests}$. Theoretically, 4603 tests are required to achieve a 99% chance of detecting moderately infrequent bugs; i.e., those which occur at a frequency of 1 in a thousand cases. [24].

Text	# of tests
Harmon & King '83 [26]	4..5
Buchanan, et. al. '83[8]	≈ 6
Babrow et.al. '86 [7]	5..10
Davies '94 [18]	8..10
Yu, Fagan et. al. '79 [46]	10
Caraca-Valente et. al. '99 [10]	< 13
Menzies '98[34]	40
Ramsey & Basili '89 [40]	50
Betta et. al. '95 [5]	200

Figure 2: Number of tests proposed by different authors. Extended from a survey by [10].

depicted in Figure 1, then over 4,500 tests are required to find moderately infrequent bugs.

However, most of the expert systems literature proposes evaluations based on very few tests[1], as indicated in Figure 2. For example, Menzies defined a minimal test procedure requiring a mere 40 successful tests to check an expert controller for a complex chemical plant (125 kilometers of highly inter-connected piping) [32]. Upon completion of testing, its performance (in terms of the correctness) is assessed via comparing it with human operators. In this procedure, 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 was compared using a t-test[2]. We were able to reject the hypothesis

[1]Exception: [3] propose at least one test for every five rules and add that "having more test cases than rules would be best".

[2]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

that the expert system performed worse than the human expert.

The core assumption in this paper is that testing samples a search space. Some parts of the space contain desired goals and some parts contain undesirable behaviour. This notion of "testing=search" covers numerous testing schemes:

- Model-checking for temporal properties [14] 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 portions of the search space that connect a literal from where it is assigned to wherever it is used [22].

- Partition testing, as defined by [24], is based on dividing the input space via a reflection over the search space looking for key literals that fork the program's conclusions into N different partitions.

- Multiple-worlds reasoners (described later in the paper) divide the accessible areas of a search space into their consistent subsets.

Based on this insight of "testing=search", the following conclusions can be inferred. Figure 1 is an accurate estimate of the number of tests if the search space inside a KBS is very complex. However, the mathematics that generated Figure 1 has no knowledge of the internal structure of a KBS. In the case where that internal structure is very simple, then Figure 2 might be a more accurate estimate of the number of tests.

The purpose of this paper is to explore the average-case structural features of expert system's search spaces. We will argue that, usually, expert systems have a simple internal structure, i.e., a KBS can be explored with a small number of tests. This conclusion will be made after comparing the effectiveness of two basic testing (search) strategies:

- The "light testing" proposes small test suites based on the few variants of the randomly selected inputs.

- The "thorough testing" reflects on how all the different KB pathways can interact and interfere with each other. Naturally, the corresponding tests suites are large and must be carefully chosen.

Based on this insight, we will define HT0, a novel, general, "light testing" strategy.

The rest of the paper is organized as follows. Section 2 gives a more detailed description of "light testing" and "thorough testing". This is followed by a literature review that indirectly supports the assumption that only a small fraction of the search space is exercised by knowledge based systems, thus resulting in small search spaces. Section 3 directly compares "light testing" with "thorough testing" using experiments

$n = m = 20$, 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 ($-t_{38,0.95} = -1.645$).

with HTx algorithms. HTx is a general testing framework for checking what percentage of desired goals can be reached across a theory. An experiment is presented, based on tens of thousands of mutants of real-world knowledge based systems. The results indicate that a very limited gain in completeness (increased coverage of the search space) can be achieved through prolonged testing. We conclude with a summary and an overview of the directions for further work in Section 4.

Note one restriction to our discussion. This paper is only concerned with the number of tests required to detect faults. A second issue, not explored here, is the number of tests required to localize and repair the detected faults. For KBS, this second issue is explored in the model-based diagnosis literature [25].

2 Literature Survey

2.1 Studies with Inference Engines

This section reviews two studies suggesting the extra effort needed for "thorough testing" would be wasted since the larger test suite would yield little more information than the smaller test suite.

Recall the informal description of the two testing strategies given in the Introduction. "Light testing" explores only a couple of randomly chosen paths while "thorough testing" considers all inputs and their downstream interactions. These two approaches to testing are represented by the following formal search strategies.

- "Thorough testing" works like an ATMS device [19]. The ATMS takes the justification for every conclusion and includes 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.

- "Light testing" is rather like a locally-guided best-first search across the internal space of the KB. When a contradiction is detected, the tester (or the search engine) could look at the contradiction for hints on how to resolve that problem. The tester (or the search engine) could then instantiate those hints and search on.

Note that, as results, the ATMS would find many options and the light searcher would only ever find one.

Williams and Nayak compared the results of their locally-guided 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 the thorough exploration of all options was nearly equivalent to the information gained from an exploration of a single option [45].

In related work, [16] compared TABLEAU, a depth-first search backtracking algorithm, to ISAMP, a randomized-search theorem prover (see Figure 3). 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;

```
Isamp(theory) {
 for TRIES := 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 3: The ISAMP randmoised-search theorem prover. [16].

	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

Figure 4: Average performance of TABLEAU vs ISAMP on 6 scheduling problems (A..F) with different levels of constraints and bottlenecks. From [16].

i.e.

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

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

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. Note that ISAMP is a "light tester" while TABLEAU is a "thorough tester" since it explores options more systematically.

Figure 4 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 easily accessable parts of a search space yielded better 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.

In summary, if most systems have this feature, then "light testing" will suffice since a few key tests are sufficient to set the control variables.

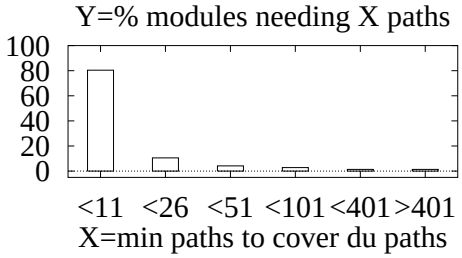

Figure 5: An analysis of hundreds of modules in a software system. From [6]

2.2 Studies with Data Sets

This section discusses two studies with data sets showing that the effective 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 a small size may yield no new information since the smaller test suite would have already explored the most frequently used parts of the KB's search space.

Avritzer et.al. studied the inputs given to an expert system for monitoring a part of the AT&T network. Each row of a state matrix stored a unique test case presented to the expert system (the combinations of literals in the given input generate one column in the matrix). After examining system usage in operational conditions (355 days of input data), they found only 857 different inputs. There were massive overlaps within this input set. On average, the overlap between two randomly selected input trajectories was 52.9%. Further, a simple analysis determined that 26 carefully selected inputs covered 99% of trajectories covered by all the observed inputs, while 53 carefully selected inputs covered 99.9% of the trajectories [2].

While Avritzer *et.al.* looked at the inputs at the system level, Colomb compared the inputs presented to an expert system with its internal structure. Recalling the introduction, the internal state space of a KB can be very large: S states per N variables implies S^N combinations. Colomb argues that the estimate of S^N is a gross over-inflation of effective size of the internal search space the KB. He argues that KBs record the very small *regions of experience* of human experts. For example, one medical expert system studied by Colomb had the theoretical size of $S^N = 10^{14}$. However, after one year's operation, the inputs to that expert system 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 ($4000 \ll 10^{14}$) [15].

In summary, if most naturally occurring test suites can be condensed like the Avritzer *et.al.* examples, then only a small number of tests will be sufficient. Further, if most systems have the Colomb small regions of expertise, then "thorough testing" is not required since "light testing" will exercise a KB's region of expertise.

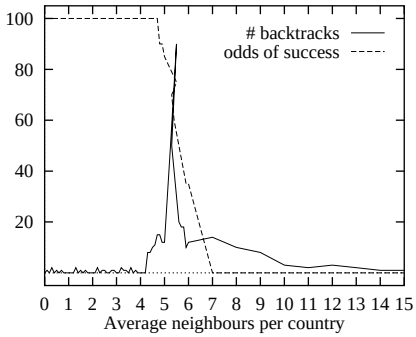

Figure 6: An example phase transition effect for the map coloring problem (no two adjacent countries on a map should use the same color). Adapted from [11].

2.3 Studies in Knowledge Engineering

One measure of the internal complexity of a program is: "how many separate execution pathways exist within that program?". Using this measure, we can say that some expert systems are not complex at all. Bieman and Schultz [6] 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*. The upper bound on the number of du-pathways in a program is exponential on the number of program statements. The lower bound on the du-pathways is 1; i.e. the tail of each path touches the head of another path.

As the du-paths shrink, it number of inputs required to reach part of the program also shrinks. Figure 5 shows the number of du-pathways in a natural language understanding program. Note that, at least for this sytem, in the overwhelming majority of their modules, very few inputs exercised all the du-pathways.

In summary, if most programs hold few du-pathways, then "light testing" will quickly explore the whole system.

2.4 Studies With Constraint Satisfaction

The previous section showed that, in one example expert system, the search space was remarkably small. This section makes a more general case: For KBS performing NP-hard tasks, most of the search space can be explored fast.

Exploring a search space is like navigating between competing constraints. Numerous recent results from the constraint satisfaction literature strongly suggest that, in the usual case, such a navigation process can be very fast.

[11] studied the runtimes of NP-hard algorithms. Figure 6 shows the typical performance. Note that outside of a narrow zone, the algorithm could terminate quickly without extensive backtracking. The slow zone corresponds to the *phase transition* between under-constrained and over-constrained problems:

- In an over-constrained problem, the odds of successfully navigating to some part of the a program is very low. Further, if the constraints are very tight, we can

quickly discover that that part of the program is unreachable since all the search options will be quickly blocked. Figure 6 is over-constrained above $X = 6$.

- In an under-constrained problem, the odds of successfully navigating to some part of a program is very high since many solutions exist. Figure 6 is under-constrained below $X = 5$.

- In the phase-transition zone, navigation can be very slow since each backtrack denotes finding a useless dead-end. The phase-transition zone must be avoided, if possible, i.e. the program should be changed. Changing the problem due to processing problems is an accepted strategy in other engineering fields. For example, mechanical engineers change the weight of a structure if its natural harmonic is close to any of the occurring environmental frequencies.

The phase-transition effect for NP-hard problems is a robust experimental result that has been replicated by numerous researchers around the world (e.g. [42, 23, 41]). Many KBS perform NP-hard tasks (e.g. diagnosis [9], planning). Testing under-constrained NP-hard KBS will quickly find the reachable program properties while tests over over-constrained NP-hard KBS will quickly fail.

In summary, if most KBS perform NP-hard tasks, then a small number of randomly chosen tests (i.e. "light testing") will yield nearly as much information as a large number of considered tests (i.e. "thorough testing").

2.5 Studies in the Software Engineering Literature

Our theme is that, on average, the search space within a KBS is far simpler than we might have thought. What happens to this theme when a KBS is coupled with a procedural system? Such hybrid implementations are very common. For example, many expert systems need support from a procedural system to handle data entry and database storage.

We argue that the addition of a procedural component to a KBS does not refute the theme of this article. Numerous results in the software engineering literature suggest that procedural systems share the "simple search space" property with KBS. When we read the source code of a procedural program, we may conceive of complex pathways that would be intricate to test. In practice, exploring these intricate pathways may not be an effective test method. Fenton and Pfleeger [21] (p302) report that we should not expect to reach all portions of the source code. In one study, even after trying to explore the entire space of a program, average reachable "objects" (paths, linearly independent paths, edges, statements) were only 40% at most. In another study, Horgan and Mathur [28] recorded the reachable objects (basic blocks) within the AWK [1] report generation language and the TEX [29] word processor. Elaborate test suites exist for those systems (e.g. [29]) yet the average coverage of TEX and AWK objects is very low, as shown in Figure 7.

In summary, parts of a procedural program may be unreachable and need not be tested. Further, the reachable portions of a program may have very simple search spaces. A *control-flow diagram* shows how one statement in a procedural program can follow from another. Given N statements in a program, the upper bound on the

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

Figure 7: Coverage of program objects, as reported in [28, 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.

number of edges in such a diagram is N^2 (i.e. every statement calls every other statement). However, in practice, the number of edges is usualy linear with respect to the number of statements [27]. That is, the structure of procedural programs looks more like simple single-parent trees than complex mazes.

If the structure of a procedural program is as simple as argued above, then a test procedure should rapidly find the reachable errors. This is often the case. Horgan and Mathur [28] observe that test suites for procedural programs often exhibit a *saturation* effect; i.e. after the first items in a test suite find many errors, further testing reveals few other errors. Saturation is consistent with simple search spaces; i.e. the reachable parts of a program can be quickly covered.

In summary, if we can easily search the reachable portions of procedural programs, as a consequence of the findings reported above, then "light testing" is sufficient to test hybrid KBS/procedural systems.

3 Experiments with HTx

The previous section discussed research from other authors that indirectly suggests we can search a KBS quickly. This section describes experiments dealing with the coverage of KBS search spaces. Based on the HTx *abductive model of testing* [33, 35], a suite of *mutators* can generate any number of sample testing problems. Informally, abduction is inference to the best explanation. More precisely, abduction makes assumptions A required to reach output goals Out across a theory T ($T \cup A \vdash Out$), without causing a contradiction ($T \cup A \nvdash \bot$). 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).

Three HTx search algorithms are used in our study. These are HT4 [33], HT4-dumb [36], and HT0 [31]. 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 [20]. HTx and QMOD assume 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. A small number of random searches (by HT0) covered nearly

as much output as extensive searches by HT4. The results endorse the widespread existence of simple search spaces.

3.1 An Informal Model of Testing

This subsection gives a quick overview of the intuitions of HTx. The next subsection 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 (make random choices, e.g. ISAMP) and so the pathway taken from inputs to outputs may vary. Unlike imperative programs, knowledge based systems must be able to come to an output even if the knowledge, provided by inputs In or the state variables (these can also be considered as a part of In) is incomplete. Hence, we call the intermediaries, corresponding to currently unknown values, the *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` with 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 reveal as much as a more thorough poking around all the corners with the flashlight.

3.2 HTx: Formal Models of Testing

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

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

- An *And* vertex can be believed if *all* its parents are believed.
- An *Or* vertex can be believed if any its parents are believed or it has been labeled an In vertex (see below). Each or-node contradicts zero 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 of type Or. A proof $P_x \subseteq G$ 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.

$$|leaves(uses(P_x))| = 1 \wedge V_i \in leaves(uses(P_x)) \wedge V_i \in Out$$

- One or more roots, which are inputs,i.e., $roots(uses(P_x)) \subseteq In$.

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

$$\forall i, j \quad < V_i, V_j > \in uses(P_x) \wedge V_j \notin no(V_i)$$

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 consists of 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 be generated by zero or more proofs $\{P_x, P_y, ...\}$. A world is a maximal consistent subset of the proofs (maximal w.r.t. size and consistent w.r.t. the no sets) denoted $proofs(W_i) = \{P_x, P_y, ...\}$. The relation from worlds W to proofs P is many-to-many. The cover of a world is expressed by the number of 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.

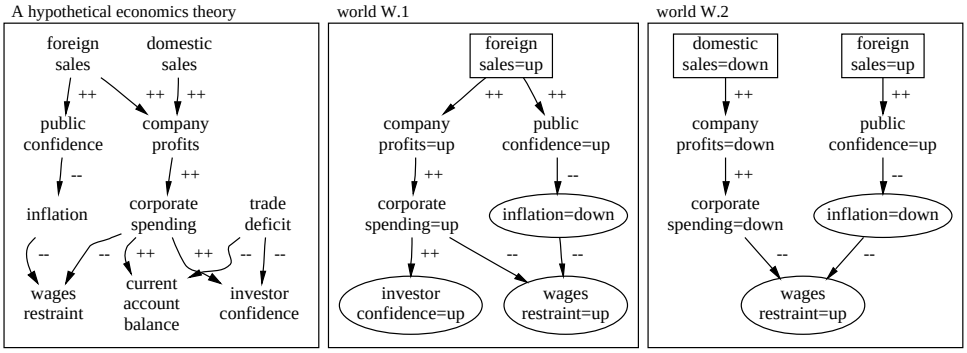

Figure 8: Two worlds for outputs (ellipses) from inputs (squares). Assumptions are world vertices that are not inputs or outputs. Note that contradictory assumptions are managed in separate worlds.

3.3 An Example

Figure 8 shows a hypothetical economics theory written in the QCM language [35]. 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, $x \overset{+\,+}{\rightarrow} y$ denotes that *y* being *up* or *down* could be explained by *x* being *up* or *down*, respectively. Also, $x \overset{-\,-}{\rightarrow} y$ denotes that *y* being *up* or *down* could be explained by *x* being *down* or *up* respectively.

Referring to the economics theory shown in Figure 8, consider the case where the inputs In are *(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 Figure 9). These proofs comprise only or-nodes. They 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 8, middle) and W_2 (Figure 8, 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%. Algorithm HT4 (and QMOD before it) found errors in a published neuroendocrinological theory [43] when its maximum cover was found to be 42%. That is, after making every assumption needed 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.

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*

Figure 9: Proofs connecting inputs to outputs. *X*†, and *X*†denote the assumptions and the controversial assumptions, respectively.

3.4 HT4, HT4-dumb, HTx

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

- HT4 finds all proofs for all outputs, then generates all the worlds from these proofs. Next, the worlds(s) with largest cover are returned.
- HT4-dumb is a crippled version of HT4 that returns any world, 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 randomized search engine that attempts to find a proof for each member of Out_i in less than `MAX-TRIES` trials. Out is explored in a random order and, while generating the proof, if more than one option is found, one of these 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 current try and discards the one with the lower cover.

Our notion of "light testing" corresponds to HT0 since it uses a quick method to study a program. "Thorough testing" is represented by HT4 since it explores more of the program. For example, if "light testing" had stumbled randomly on W_2 of Figure 8, and did not look any further to find W_1. Then we would have inappropriately declared that the economics model could only explain 67% of the outputs. However, on average, the cost of the extra searching provided by "thorough testing" may not be justified by its benefits. The following studies, comparing HTx algorithms, support the case for using "light testing".

3.5 HT4 vs HT4-dumb

We compared HT4 and HT4-dumb using tens of thousands of theories. Starting with a seed theory (fish growing in a fishery, see Figure 10), automatic *mutators* were built to generate a wide range of related problems. The mutators used in this work are shown in Figure 11. Each mutator took a known real-world problem and distorted it. The mutators were designed so that the distortions were "stepped"; i.e.:

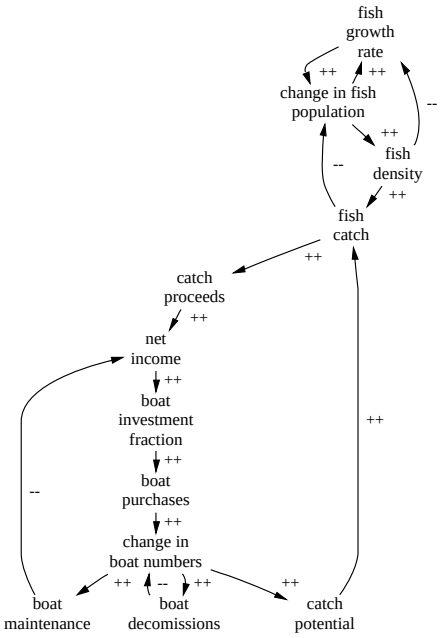

Figure 10: The QCM fisheries model contains two $\frac{dX}{dT}$ variables: *change in fish population* and *change in boat numbers*.

- An application of a mutator generated a problem that was nearly like the original problem;
- A several subsequent applications of mutators generated problems that were very different from the original problem.

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 12. 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.

3.6 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, the "fast", randomized search engine. 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 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)

1. Add influences between variables.
2. Corrupt the influence between two variables; e.g. flip proportionality `++` to inverse proportionality `--` or visa versa.
3. Experiment with different meanings of time within the system. In the *explicit node* (XNODE) interpretation of time, all $\frac{dX}{dT}$ variables imply 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 imply a link $(x\ at\ Time^i) \stackrel{\alpha}{\rightarrow} (y\ at\ Time^{i+1})$; $(\alpha \in \{++, --\})$. XNODE and IEDGE are contrasted in Figure **??**. Why these two interpretations of time? These were randomly selected from a much larger set of time interpretations studied by [44].
4. Force 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 $U = 100 - \frac{(|In|+|Out|)*100}{|V|}$ and $|V|$ represents the number of variables in the theory. The mutators build In and Out sets with different U settings. Values for In and Out are collected using a mathematical simulation of the fisheries.

Figure 11: Problem mutators used in the HT4 vs HT4-dumb study

curve of Figure 13 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 for different values of `MAX-TRIES` was explored. Surprisingly, the maximum cover found at `MAX-TRIES=50` was identified much earlier, when `MAX-TRIES` was 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 search space as a careful, slower, larger number of searches (HT4).

4 Conclusion

On average, it appears that exploring a theory is not as complex as one might think. We defined the testing of knowledge based systems as a search process and a test suite is considered complete when the search achieves desired coverage. We have experimentally demonstrated that a few explorations of a search space yield 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 rigorously. 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.

The implications of the observed search phenomenon for testing of knowledge based systems are important. Simple random strategies for testing of knowledge based

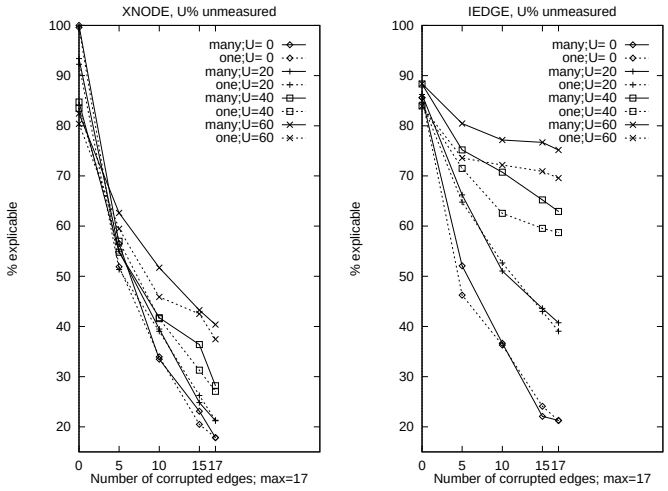

Figure 12: Comparing coverage of Out seen in 1,512,000 runs of the HT4 (solid line) or HT4-dumb (dashed line) algorithms with different percentages of unmeasured U variables. X-axis refers to how many edges of Figure 10 were corrupted (`++,--` flipped to `--,++` respectively.)

systems appear to be as powerful as the more detailed (and much slower) testing strategies. While there still exists the trade-off between the achieved coverage and the size of the test suite, the data collected from the experiments indicate that substantial coverage may be achieved by test suites containing a surprisingly small number (few dozens) of randomly chosen tests.

Our future plans include expanding the results of this study by estimating the reliability of knowledge based systems (see [30] for some preliminary results). It appears that incorporation of coverage measures into the reliability estimation of knowledge based systems may save significant resources (time and effort) in comparison with the black box approach, for example. Admittedly, the naive interpretation of testing results could lead to an overly optimistic reliability prediction. The results of this article refer to the average case behaviour of a test suite. In safety-critical situations, such an average case analysis would be inappropriate due to the disastrous implications of the non-average case.

Acknowledgements

This work was partially supported by NASA through cooperative agreement #NCC 2-979, and through the Research Initiation grant to one of the authors.

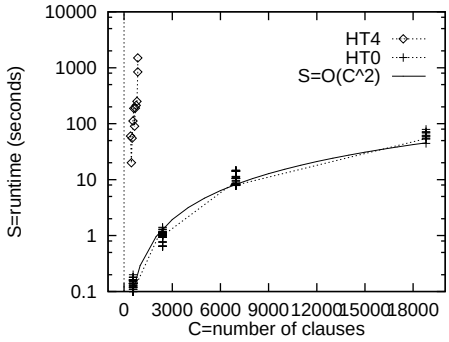

Figure 13: Runtimes HT4 (top left) vs HT0 (bottom). HT0 was observed to be $O(N^2)$.

References

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

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

[3] 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.

[4] F. Bastani and I. Chen. Assessment of the reliability of ai programs. *International Journal of Software Engineering and Knowledge Engineering*, 3(1):99–114, 1993.

[5] 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.

[6] 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.

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

[8] 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.

[9] T. Bylander, D. Allemang, M.C. M.C. Tanner, and J.R. Josephson. The Computational Complexity of Abduction. *Artificial Intelligence*, 49:25–60, 1991.

[10] 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.

[11] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-91*, pages 331–337, 1991.

[12] I. Chen. Effect of parallel planning on the system reliability of real-time expert systems. *IEEE Transactions on Reliability*, 46(1):81–87, 1997.

[13] I. Chen and T. Tsao. A reliability model for real-time rule-based expert systems. *IEEE Transactions on Reliability*, pages 54–62, March 1995.

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

[15] 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.

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

[17] A. Noor D. Goldin, S. Venneri. A new frontier in engineering. *Mechanical Engineering*, pages 62–69, February 1998.

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

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

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

[21] N. E. Fenton and S.L. Pfleeger. *Software Metrics: A Rigorous & Practical Approach*. International Thompson Press, 1997.

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

[23] I. Gent, E. MacIntyre, P. Prosser, and T. Walsh. Scaling effects in csp phase transistion. In *International Conference on Principles and Practice of Constraint Programming*, 1995.

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

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

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

[27] 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.

[28] 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.

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

[30] T. Menzies, B. Cukic, and E. Coiera. Smaller, faster dialogues via conversational probing. In *AAAI'99 workshop on Conflicts and Identifying Opportunities. Available from* http://research.ivv.nasa.gov/docs/techreports/, 1999.

[31] 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://research.ivv.nasa.gov/docs/techreports/1999/NASA-IVV-99-007.pdf, 1999.

[32] Tim Menzies. Critical success metrics: Evaluation at the business-level, 2000. International Journal of Human-Computer Studies, special issue on evaluation of KE techniques.

[33] T.J. Menzies. *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales. Avaliable from http://www.cse.unsw.edu.au/~timm/pub/docs/95thesis.ps.gz, 1995.

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

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

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

[37] T.A. Nguyen, W.A. Perkins, T.J. Laffey, and D. Pecora. Knowledge base verification. *AI Magazine*, 8(2):69–75, 1987.

[38] A.D. Preece. Principles and practice in verifying rule-based systems. *The Knowledge Engineering Review*, 7:115–141, 2 1992.

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

[40] 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.

[41] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *AAAI '92*, pages 440–446, 1992.

[42] B. Smith and M. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, (81):155–181, 1996.

[43] G.A. Smythe. Brain-hypothalmus, Pituitary and the Endocrine Pancreas. *The Endocrine Pancreas*, 1989.

[44] S. Waugh, T.J. Menzies, and S. Goss. Evaluating a qualitative reasoner. In Abdul Sattar, editor, *Advanced Topics in Artificial Intelligence: 10th Australian Joint Conference on AI*. Springer-Verlag, 1997.

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

[46] 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.

[47] N. Zlatereva and A. Preece. State of the art in automated validation of knowledge-based systems. *Expert Systems with Applications*, 7:151–167, 2 1994.