Many Maybes Mean (Mostly) the Same Thing

Tim Menzies[1], Harhsinder Singh[2]

[1] Lane Department of Computer Science
West Virginia University; tim@menzies.com
[2] Department of Statistics, West Virginia University
hsingh@stat.wvu.edu

Abstract. At the core of soft computing is the intuition that from imprecise knowledge, we can still make reasonable inferences. This paper offers experimental and mathematical evidence for this intuition. Based on a literature review and a newly developed mathematics of "reachability", it is argued that searches through a space containing uncertainties, most of the reachable conclusions will be reached via a small number of "master variables" in a "narrow funnel". Such narrow funnels can be found using very simple randomized search methods.

1 Introduction

Some intuitions, while compelling, may not be correct. For example, consider Zadeh's intuition that:

> *... as the complexity of a system increase, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive properties.*
>
> –Lofti Zadeh[48]

Our own pre-experimental notions was that this intuition was essentially correct. It seemed clear that the more we say, the less we are certain on what we say. As theory complexity increases, the certainty of that theory's assertions decreases as we struggle to fill details which we may never have explored before. One way in which complex theories get imprecise is the presence of "many maybes"; i.e. multiple points where it is unclear which mutually incompatible assertion should be made. This may be as simple as a dispute between different designers over the size of a numeric constant in an equation. Alternatively it may be as complex as a qualitative reasoner that generates innumerable possible conclusions, one for each set of consistent possibilities within a large space of contradictions. In either case, the problem is the same: assertions about some point are contradictory.

However, after reviewing the available evidence, these intuitions must be revised. A repeated observation is that within the current generation of software, *many maybes*

[0] An earlier version of this paper, with the same title, appeared in the 2nd International Workshop on Soft Computing applied to Software Engineering, Netherlands, February, 2001: http://varlet.csc.uvic.ca/~scase01/

mostly mean the same thing. That is, if we ask software containing contradictory assertions to report on all the ways we achieve certain goals, then there emerge certain goals that are always true, or always false, across the wide space of "maybes". For these stable inferences, we can make precise and categorical statements in the presence of complex and possible uncertain assertions. Suppose these experimental observations are a general result, and not just a result of a quirky selection of case studies. If so then we have an explanation for the success of fuzzy logic [48], genetic algorithms [13], neural nets [44], qualitative reasoning [25], heuristic programming [7], stochastic inference (e.g. ant intelligence [16], ISAMP[14], HT0 [37] GSAT [43], black-box testing [21]) and many other approximate soft reasoning techniques. These techniques work not because of their intrinsic power, but because many probes across a space of uncertainties will achieve the same result.

This chapter tests the generality of the experimental observation that many maybes mostly mean the same thing. At issue is how much effort we should spend on the construction of elaborate soft computing tools. We will offer experimental and theoretical evidence that, in the general case, very simple tools such as the random search of HT0 (described in §3) or the limited learning of TAR2 (described in §6) should suffice for soft computing tasks that reason about a space of uncertainties.

The theoretical case that many maybes mean mostly the same thing is based on the "funnel theory" of Menzies, Easterbrook, Nuseibeh and Waugh [32]. Funnel theory, as presented in §2, has an intuitive appeal and explains the counter-intuitive experimental observations listed in §3. However, until this chapter, funnel theory had no formal basis. Based on a mathematical argument, it will be shown that we can routinely expect our software to contain narrow funnels. This maths will be presented in two parts. Firstly, in §4, an average case *reachability* model is presented that computes the odds of reaching some randomly selected part using a theory that contains contradictions. This model has an odd behavior: the number of contradictions per literal does not greatly effect the output of the model. This odd behavior prompted the development a second model. Based on a simulation of an abstract model of funnels, §5 argues that if some conclusion can be reached via a narrow funnel and a wide funnel, then a random search will tend to use the narrower funnel. The argument is recursive: given a narrow funnel and a narrower funnel, random search will favor the narrower funnel over the narrow funnel. Hence, the narrowest funnels act like strange attractors in chaos theory, pulling in all the arguments. Since these arguments will use narrow funnels, there will be few points of disagreement. Hence, the net result of most of the disagreements will be very similar that most maybes will mean the same thing. §6 presents TAR2: an application of TAR2 in which a very simple search device is adequate for controlling a diverse range of devices. TAR2 is such a dumb algorithm that its repeated success is inexplicable *unless* narrow funnels are common.

2 Funnel Theory

According to funnel theory, arguments within software are pathways through a space of possible contradictions. Each pathway leads to some desired goals and contains a set of assignments to variables. Given a set of goals, if we build proof trees for each

goal separately, then it is possible that these proofs will demand different assignments to the same variables. That is, the proofs are contradictory around those variables. The set of variables with contradictory assignments are called the *funnel* of an argument. The cardinality of this set is a measure of how much the conclusions from this theory can vary. Given and S arguments about the assignments to N variables in the funnel, then there are S^N combinations of proof trees that we can believe at the same time. Depending on which assignment we endorse, different proof trees will be endorsed and different goals will be reachable.

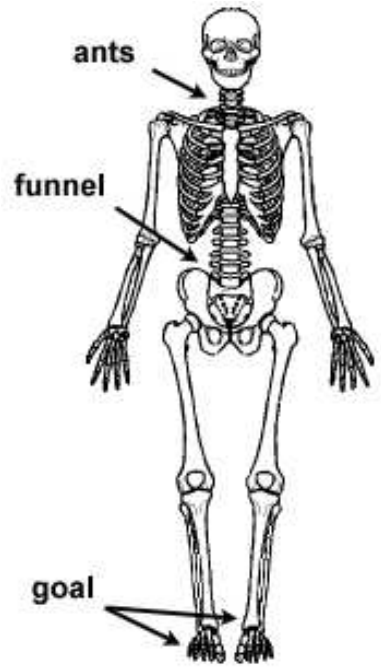

Fig. 1.

As the funnel size N shrinks, then there are exponentially less different ways to resolve the contradictions in a theory and exponentially less methods for reaching different goals. Funnel theory claims that most searches through a space of contradictory options will lead to the same goals if the pathways cross *very narrow funnels*. Narrow funnels have two properties suggesting that many maybes will lead to the same consequences. Firstly, narrow funnels dictate how arguments must be resolved around the funnel. If an argument *must* make it through a funnel in order to reach a goal, then that argument must adapt itself to the shape of the funnel. Secondly, narrow funnels let us ignore certain disagreements. Consider two arguments: one around a narrow funnel and another very peripheral to that funnel. The funnel argument could be resolved quickly since only certain resolutions will pass through the funnel. Further, we need not spend

much time on the peripheral argument since it is likely that most pathways will never use that peripheral part of the model.

To understand the effects of funnels consider some ants at the neck of Figure 1 arguing about how to best crawl down to the feet. Each ant's argument relates to one possible pathway across the skeleton. Note that our search space has funnels: all the pathways must pass through the lumbar spine just above the hips. Our ants might have different disputes about the best way to handle fingers, ribs, and the lumbar spine. Some of these arguments are irrelevant. For example, arguing about how to traverse a finger is irrelevant to the goal of reaching the feet since no pathway through the fingers takes us to the ground without returning to our current position at the neck. Also, with respect to some stated goal, the presence of funnels ensures that some of these arguments only have one possible resolution. For example, suppose one of our ants prefers not to crawl around the lumbar spine since the bones there are too pointy. Given the goal of vertical motion to the feet across the skeleton, that ant must surrender to the inevitable and travel across the pointy lumbar spine. Clearly, if the software contains the same narrow funnels as Figure 1, then we would expect that the net effect of the contradictory possibilities within that software would be the same.

3 Experimental Evidence

This section reviews empirical evidence that narrow funnels are common in software systems. Elsewhere, Menzies and Cukic [31] have cataloged the number of tests used to certify expert systems. In theory, probing a space to find a bug with probability 10^{-x} takes $4.6 * 10^x$ tests to be 99% sure of finding that event. To show this, note that N randomly selected inputs has certainty

$$C = 1 - \left((1-x)^N\right)$$

of finding some event with probability x. Hence, at a 99% certainty, $C = 0.99$ and this equation becomes:

$$N = \frac{ln(1-0.99)}{ln(1-x)} = \frac{-4.6}{ln(1-x)} \quad (1)$$

If the space is being probed by a nondeterminate search engine, as often used in a

reference	[22]	[6]	[5]	[15]	[47]
# tests	4..5	≈ 6	5..10	8..10	10
reference	[9]	[40]	[42]	[3]	
# tests	< 13	40	50	200	

Fig. 2. Number of tests used to certify expert systems.

heuristic-based expert system, then $4.6 * 10^x$ would be a theoretical lower bound on the required number of tests. Nevertheless an often repeated observation is that a small

number of inputs can often reach significant errors in a program (see Figure 2). One explanation for this surprising observation is that narrow funnels very quickly drive a small number of test cases towards the reachable failures.

Similarly, in conventional software, surprisingly few random probes will detect significant errors in a system. Leveson heuristically applied partial descriptions of software fault trees to ADA code. She claims that this heuristic search detected as many errors in her studied system as a much longer, and much more formal, analysis [27]. If conventional software contained narrow funnels, that would explain how Leveson's heuristic partial probing was so successful since any argument, generated either by formal or informal methods, would both be sucked towards the funnels.

Another method of probing a system is mutation testing. In mutation testing, a test suite is assessed via its ability to distinguish a program from some mutation of that program. Numerous researchers in mutation testing report that most mutations give rise to the same nominal and off-nominal behaviors [8, 41, 46, 2]. This result can be explained assuming narrow funnels. Mutators are applied randomly and if the funnels are small, it is unlikely that the mutators will stumble across them.

Another reason to believe in narrow funnels is that the overall structure of our programs may not support wide chains of arguments. Bieman and Schultz [4] report that a seemingly complex natural language processing system contains, on average, a small number of narrow *du-pathways*. A du-path is a link from where a variable is *defined* to where it is *used*. Clearly, 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. Figure 3 shows the Bieman and Schultz results: 95.1% of the modules in their system held less that 50 du-pathways. Analogous results has been seen in procedural code. In one analysis of 4000 FORTRAN functions and 3147 "C" functions, the control flow graph of those functions grows linearly with the number of statements [23]. That is, the control-flow diagram forms a single-parent tree. Arguments extracted from single-parent trees would be very narrow indeed.

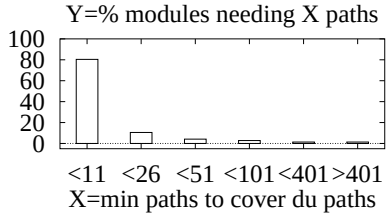

Fig. 3. Path found in software modules by [4]

There is much evidence that the average size of a funnel in AI-based systems is very narrow. Researchers in AI and requirements engineering explore inconsistent theories. A repeated result consistent with narrow funnels is that committing to a randomly selected resolution to a conflict reaches as much of a program as carefully exploring all

resolutions to all conflicts. For example, Figure 5 shows Crawford and Baker's [14] comparison of a standard depth first search backtracking algorithm (TABLEAU) to ISAMP, a randomized search theorem prover (shown in Figure 4). ISAMP randomly assigns a value to one variable, then infers some consequences using a fast forward chainer. After forward chaining, if incomparable conclusions were reached, ISAMP reassigns all the variables and tries again (giving up after `MAX-TRIES` number of times). Otherwise, ISAMP continues looping till all variables are assigned. When implemented, Crawford and Baker found that ISAMP took *less* time than TABLEAU to reach *more* scheduling solutions using, usually, just a small number of `TRIES`. 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. Note that this dependent-control theory is consistent with narrow funnels: the small number of control variables are those found in the narrow funnels. TABLEAU failed since it's rigorous search demanded the resolution of unimportant arguments outside the narrow funnels.

```
for TRIES := 1 to MAX-TRIES
 {set all vars to unassigned;
  loop
   {if   everything assigned
    then  return(assignments);
    else  pick any var v at random;
          set v's value at random;
          forwardChainFrom(v);
          if   contradiction
          then exit loop;
          fi
    fi
   }
  } return failure
```

Fig. 4. The ISAMP algorithm [14]

Experiments with randomized multiple-worlds inference engines support the thesis that narrow funnels are common. If multiple worlds of belief are created via very wide funnels, then we would expect a large number of worlds created with each world condoning possibly different inferences. However, if such worlds are created through narrow funnels, then we would see that only a small number of worlds are created. Further, since there are few disagreements between the worlds, it is likely that the created worlds would condone similar inferences. In results consistent with narrow funnels, Menzies, Easterbrook, Nuseibeh and Waugh [32] report that exploring a few set-covering worlds returned nearly as much information as a rigorous exploration of all worlds. That experiment is described below.

The Menzies, Easterbrook, Nuseibeh and Waugh study compared the behavior of

	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

Fig. 5. Average performance of elaborate search (TABLEAU) vs randomized search (ISAMP) on 6 scheduling problems (A..F) with different levels of constraints and bottlenecks.

two multiple-world reasoners: HT0 and HT4. HT4 generates all pathways from inputs to goals and sorts them into consistent worlds of belief [39]. HT0 just returns the first world it finds randomly [37]. HT0 randomizes the order in which it searches for proofs. During the proof of goal i, when processing a set of goals in a disjunction or a conjunction, the order of the processing is selected randomly (see `rand/2, ror/2` in Figure 6). If a proof of goal i fails, the system does not backtrack to retry one of goal $1 \ldots (i-1)$. Instead, HT0 lowers a weight associated with goal i and moves on to try goal $i+1$ (see `prove/2` in Figure 6). When HT0 has finished with all the goals, it wipes all the assumptions, sorts the goal list according to the adjusted weights, then tries to prove them all again (see `ht0/2` in Figure 6). When HT0 and HT4 were run on the same examples, HT4's runtimes were observed to be exponential while HT0 was less than cubic [37]. Also, and most important for our discussion, the random search of HT0 reaches nearly as many goals as the rigorous search of HT4. Menzies, Easterbrook, Nuseibeh and Waugh executed thousands of models using HT0 and HT4. To generate these models, mutators would corrupt influences in a theory; e.g. proportional signs were flipped to inversely proportional and visa versa, influences were added at random, and less and less data was offered to the reasoner. In a result consistent with most maybes mean the same thing, the average difference in covered goals between the random partial search of HT0 and the rigorous search of HT4 was less than 6% (see Figure 7).

4 Generalizing HT0 with Reachability Theory

Did HT0 work because of quirks in its case study? Or was it an example of a general principle? This section argues that HT0's results are quite general: the average odds of reaching a goal across a space containing contradictions is quite high. These average-case odds can be calculated using the *reachability analysis* [30] described below.

Reachability studies the odds of reaching a random part of *NAYO graphs* like Figure 8. Such NAYO graphs contain *No-edges*, *And-nodes*, *Yes-edges*, and *Or-nodes*. Yes-edges denote valid inferences and no-edges denote incompatibilities "maybes".

```
%test 'ors' in a random order
X ror  Y :- maybe -> (X;Y); (Y;X).

%test 'ands' in a random order
X rand Y :- maybe -> (X,Y); (Y,X).

maybe    :- 0 is random(2).

% Assuming that an object O's attribute
% A is X  is legal if this assumption does
% not conflict  with previous assumptions.
% Otherwise, make  assume that O.A=X but
% remove it if ever we  backtrack to this point.
A of O is X :- a(A,O,Old), !, Old = X.
A of O is X :- assert(a(A,O,X)).
A of O is X :- retract(a(A,O,X)), fail.

% N times, zap assumptions, try the goal list.
ht0(0,_)   :- !.
ht0(N0,G0) :- rememberBestCover(G0),
              retractall(a(_,_,_)),
             % Goals with lower weights
             % are tried first
              sort(G0, G1),
              maplist(prove,G1,G),
              N is N0 - 1,
              ht0(N,G).

% Lower/raise a goal's weight by a random amount
% if it fails/works  respectively.
prove(In/Goal,Out/Goal):-
              X is 1 + random(10^3)/10^6,
              (call(Goal) -> Out is In+X; Out is In-X).

% E.g: 5 times,  random search for "sad" or "rich".
:- ht0(5,[1/sad,1/rich]).
```

Fig. 6. HT0, simplified (handles acyclic ground theories only). The full version contains many more details such as how variables are bound within `rand`s and the implementation of `rememberBestCover`. For full details, see [37].

The V nodes of a NAYO graph are divided into and-nodes and or-nodes with ratios $andf$ and orf respectively ($orf + andf = 1$). In the reachability model, and-nodes and or-nodes have mean parents $andp, orp$ respectively. Or-node contradict, on average, no other or-nodes. $andp, orp, no$ are random gamma variables with means $andf_\mu, andp_\mu, orp_\mu, no_\mu$; "skews" $andp_\alpha, orp_\alpha, no_\alpha$; and range $0 \leq \gamma \leq \infty$. $andf$ is

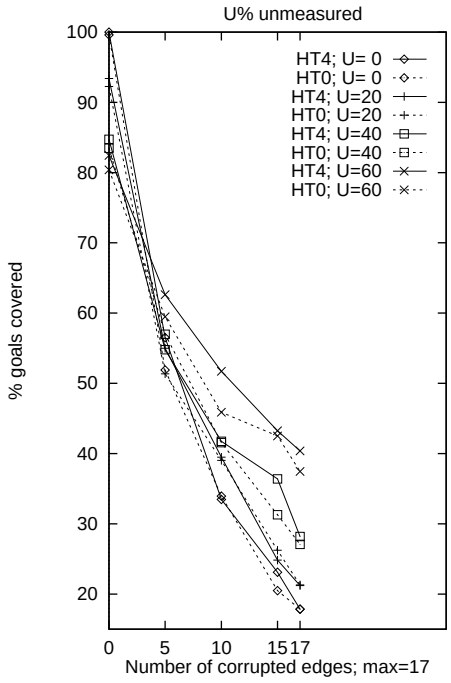

Fig. 7. HT4 (solid line) vs HT0 (dashed line). $U\%$ denotes what percentage of the available domain data was ignored in each run.

a random beta variable with mean $andf_\mu$ and range $0 \leq \beta \leq 1$. And-nodes are reached at height j via one parent at height $i = j - 1$ and all others at height:

$$i = \beta(depth) * (j - 1) \tag{2}$$

so $0 \leq i \leq (j - 1)$. Note that as $depth$ decreases, and-nodes find their pre-conditions closer and closer to the inputs.

The probability $P[j]_{and}$ of reaching an and-node at height $j > 0$ is the probability that one of its parents is reached at height $j - 1$ and the rest are reached at height $1..(j - 1)$; i.e.

$$P[j]_{and} = P\left[j - 1\right] * \left(\prod_{2}^{andp[j]} P[i] \right) \tag{3}$$

Or-nodes are reached at height j via one parent at height $i = j - 1$. The probability $P[j]_{or}$ of reaching an or-node at height $j > 0$ is the probability of not missing any of its parents; i.e.

$$P[j]_{or} = 1 - (1 - P\left[j - 1\right]) * \left(\prod_{2}^{orp[j]} (1 - P[i]) \right) \tag{4}$$

```
diet(fatty).
diet(light).
happy  :-
   tranquillity(hi)
   ; rich,healthy.
healthy  :- diet(light).
satiated :- diet(fatty).
tranquillity(hi) :-
   satiated
   ; conscience(clear).
```

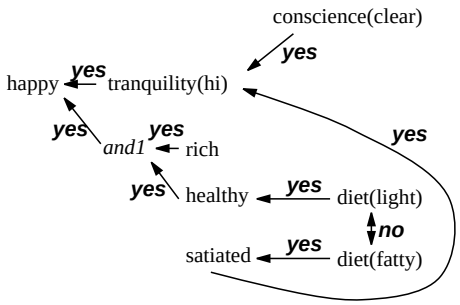

Fig. 8. A NAYO graph (shown right) connecting terms within some theory (shown left).

From $P[j]$, we compute $P'[j]_{or}$ by modifying $P[j]$ with two factors: one for the odds of not entering into an inferencing loop, and one for the odds of not causing contradictions:

$$P[j]_{no\,contradiction} = \left(1 - \frac{no}{V}\right)^{n[j]*orf} \tag{5}$$

$$P[j]_{no\,loop} = \left(1 - \frac{1}{V}\right)^{n[j]*orf} \tag{6}$$

where $n[j]$ is a guesstimate of the size of the proof tree to depth j. Observe the use of $n[j]*orf$ in Equation 5 and Equation 6. And-nodes contradict no other nodes; hence we only need to consider contradictions for orf of the system. Also, since every and-node has an or-node as a parent, then we need only check for loops amongst the or-nodes.

The probability $P[j]$ of reaching any node is hence the sum of $P'[j]_{or}$ and $P[j]_{and}$ weighted by the frequencies of and-nodes and or-nodes; i.e.

$$P[j] = andf * P[j]_{and} + orf * P'[j]_{or} \tag{7}$$

$$P'[j]_{or} = P[j]_{or} \ * \ P[j]_{no\,loop} \ * \ P[j]_{no\,contradiction} \tag{8}$$

A simulation of the above system of equations is around 200 lines of Prolog. This model can be executed to generate $P[j]$. From this figure, we find the number of tests N required to be $C = 99\%$ percent certain of reaching a random node in a dependency graph using Equation 1.

The above model was run for a wide range of input parameters; e.g. up to 10^8 nodes, up to 1000 inputs, wildly varying the frequency and skew of and-nodes, or-nodes, and no-edges, etc. The frequency distribution of the generated N values is shown in Figure 9 divided according to the j (proof height) value. The simulation results shows that HT0's success was not a quirk of the models in its domains. Rather, if we explore a NAYO graph to more than a shallow depth ($j > 50$) then in the usual case, we can reach most parts of that theory with small number of random inputs.

5 Formal Funnel Theory

A formal analysis of funnel theory explains why the odds of reaching some randomly selected part of a theory is barely effected by the number of contradictions in that theory. In this section, a mathematical simulation demonstrates that given the choice of a narrow or a wide funnels to reach a goal, a random search engine will select the narrow funnel. That is, even if a theory supports many arguments, randomized search will favor the less contentious parts of a theory.

Suppose some goal can be reached by a narrow funnel M or a wide funnel N as follows:

$$\left.\begin{array}{r} \xrightarrow{a_1} M_1 \\ \xrightarrow{a_2} M_2 \\ \cdots \\ \xrightarrow{a_m} M_m \end{array}\right\} \xrightarrow{c} goal_i \xleftarrow{d} \left\{\begin{array}{l} N_1 \xleftarrow{b_1} \\ N_2 \xleftarrow{b_2} \\ N_3 \xleftarrow{b_2} \\ N_4 \xleftarrow{b_2} \\ \cdots \\ N_n \xleftarrow{b_n} \end{array}\right.$$

Under what circumstances will the narrow funnel be favored over the wide funnel? More precisely, when are the odds of reaching $goal_i$ via the narrow funnel much greater that the odds of reaching $goal_i$ via the wide funnel? To answer this question, we begin with the following definitions. Let the M funnel use m variables and the N funnel use n variables. Each member of M is reached via a path with probability a_i while each

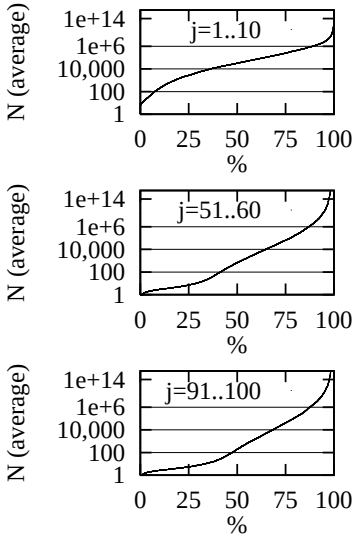

Fig. 9. Some frequency distributions of the number of tests required to be 99% sure of reaching a node at height j generated from the Menzies-Cukic-Singh reachability model.

member of N is reached via a path with probability b_i. Two paths exist from the funnels to this goal: one from the narrow neck with probability c and one from the wide neck with probability d. The probability of reaching the goal via the two pathways is:

$$narrow = c \prod_{i=1}^{m} a_i \tag{9}$$

$$wide = d \prod_{i=1}^{n} b_i \tag{10}$$

For comparison purposes, we express the size of the wider funnel as a ratio α of the narrower funnel; i.e.

$$n = \alpha m \tag{11}$$

Assuming that the goal is reached, then there are three ways to do so. Firstly, we can reach the goal using both funnels:

$$narrow \wedge wide = narrow.wide \tag{12}$$

Secondly, we can reach the goal using the narrow funnel and not the wider funnel:

$$narrow \wedge \neg wide = narrow(1 - wide) \tag{13}$$

Thirdly, we can reach the goal using the wider funnel and not the narrow funnel.

$$\neg narrow \wedge wide = (1 - narrow)wide \tag{14}$$

Let g be probability of reaching $goal_i$. Clearly, g is the sum of Equation 12, and Equation 13, Equation 14; i.e.

$$g = narrow + wide - narrow.wide \tag{15}$$

Given the goal is reached, then the conditional probabilities of reaching the $goal_i$ via two our funnels is:

$$P\left(narrow|g\right) = \frac{narrow}{narrow + wide - narrow.wide} \tag{16}$$

$$P\left(wide|g\right) = \frac{wide}{narrow + wide - narrow.wide} \tag{17}$$

The odds of an event with probability $P(X)$ is the ratio of that event to it's complement; i.e. $\frac{P(X)}{1-P(X)}$. Hence, the odds of Equation 16 is:

$$\begin{aligned} Odds\left(narrow|g\right) &= \frac{\frac{narrow}{narrow+wide-narrow.wide}}{1 - \left(\frac{narrow}{narrow+wide-narrow.wide}\right)} \\ &= \frac{narrow}{wide\left(1 - narrow\right)} \end{aligned} \tag{18}$$

Similarly, the odds of Equation 17 is:

$$Odds\left(wide|g\right) = \frac{wide}{narrow\left(1-wide\right)} \tag{19}$$

We divide Equation 18 by Equation 19 to compute the ratio R of the conditional odds of reaching $goal_i$ via the narrow or the wide funnel:

$$R = \frac{\left(narrow\right)^2\left(1-wide\right)}{\left(wide\right)^2\left(1-narrow\right)} \tag{20}$$

Our pre-condition for use of the narrow funnel is:

$$R > 1 \tag{21}$$

In general, using the narrow funnel is much more likely if R is very large, i.e. bigger than some threshold value t

$$R > t \tag{22}$$

where t is some number much larger than 1.

We can now define a procedure for finding situations when a random search engine will favor narrow funnels over wide funnels:

- For a wide range of values of $a_i, b_i, c, d, m, \alpha, \ldots$
- Look for situations when Equation 22 is satisfied.

We apply this procedure below, twice:

- In the first application, we make some simplifying assumptions such as a_i and b_i come from uniform probability distributions. These simplifying assumptions let us derive expressions for the ratios of c and d that would satisfy Equation 22.
- In the second application, we reject the simplifying assumptions and describe a simulation that handles a wider range of cases.

In both applications, it is clear that if we grow the wide funnel wider, then Equation 22 is often satisfied.

5.1 The Uniform Case

Consider the simple case that a_i and b_i come from uniform probability distributions, i.e.

$$\begin{aligned} \sum_{i=1}^{m} a_i &= 1 \\ \therefore a_i &= \frac{1}{m} \\ \therefore narrow &= c\left(\frac{1}{m}\right)^m \text{ (using Equation 9)} \end{aligned} \tag{23}$$

Similarly

$$wide = d\left(\frac{1}{n}\right)^n \text{ (using Equation 10)} \tag{24}$$

Thus, by Equation 21, narrow funnel is more likely when:

$$narrow^2(1 - wide) > wide^2(1 - narrow)$$

which we can rearrange to

$$(narrow - wide)(narrow + wide - narrow.wide) > 0 \tag{25}$$

Equation 25 contains two terms, the second of which is Equation 15 which is always positive. Hence, Equation 25 is positive when $\frac{narrow}{wide} > 1$. Substituting in Equation 23 and Equation 24, yields:

$$\frac{narrow}{wide} = \frac{c\left(\frac{1}{m}\right)^m}{d\left(\frac{1}{n}\right)^n} \tag{26}$$

Recall that $n = \alpha m$, i.e. Equation 26 will hold when:

$$(\alpha m)^{\alpha m} m^{-m} > \frac{d}{c} \tag{27}$$

Consider the case of two funnels, one twice as big as the other; i.e. $\alpha = 2$. Equation 27 can be rearranged to show that $\frac{narrow}{wide} > 1$ is true when

$$(4m)^m > \frac{d}{c} \tag{28}$$

At $m = 2$, Equation 28 becomes $d < 64c$. That is, to access $goal_i$ from the wider funnel, the pathway d must be 64 times more likely than the pathway c. This is not highly likely and this becomes less likely as the narrower funnel grows. By the same reasoning, at $m = 3$, to access $goal_i$ from the wider funnel, the pathway d must be 1728 times more likely than the narrower pathway c. That is, under the assumptions of this uniform case, as the wide funnel gets wider, it becomes less and less likely that it will be used.

5.2 The Non-Uniform Case

We have seen that the two assumptions of

1. low threshold value of $t = 1$ and
2. uniform probability distributions for the funnel preconditions

means that the narrow funnel is far more likely than the wider funnel. This section relaxes these two assumptions to use very large values of t and wildly varying values for a_i and b_i. A small simulator is used to compute Equation 22 as follows. The mean μ and standard deviation σ of the logarithm of the variables a_i, b_i, c, d were picked at random from the following ranges:

$$\mu \in \{1, 2, \ldots 10\} \tag{29}$$
$$spread \in \{0.05, 0.1, 0.2, 0.4, 0.8\} \tag{30}$$

μ and $spread$ where then converted into probability as follows:

$$\sigma = spread * \mu$$
$$probability = 10^{-1 * normDist(\mu,\sigma)} \tag{31}$$

Note that this method produces non-uniform probabilities for a_i and b_i. Next, m and α were picked at random from the ranges:

$$m \in \{1, 2, \ldots 10\} \tag{32}$$
$$\alpha \in \{1, 1.25, 1.5, \ldots 10\} \tag{33}$$

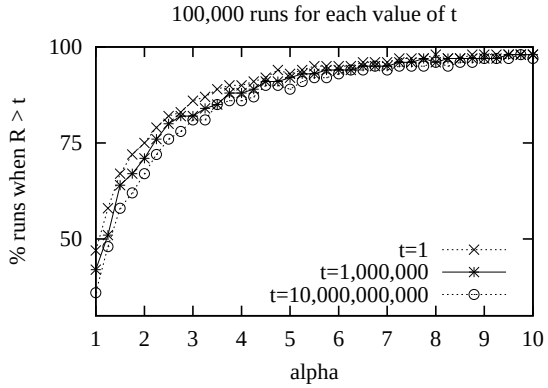

Fig. 10. Outputs from 100000 runs of the funnel simulator. The Y-axis shows what percentage of the runs satisfies Equation 22 as α increases. On the plot, α is shown as "alpha".

R was then calculated and the number of times R exceeded different values for t is shown in Figure 10. As might be expected, at $t = 1, \alpha = 1$ the funnels are the same size and the odds of using one of them is 50%. As α increases, then increasingly Equation 22 is satisfied and the narrower funnel will be preferred to the wider funnel. The effect is quite pronounced. For example, in 82% of our simulated runs, random search will be 10,000,000,000 times as likely as to use funnels $\frac{1}{3}$ smaller than alternate wider funnels (see the $\alpha = 3$ results).

In summary, in both the uniform and non-uniform case, many maybes mostly mean the same thing. Perhaps the reason for this is as a funnel widens, it becomes exponentially less likely that a random search engine will find all the members of the wider

funnel. What ever the underlying cause, the effect is clear: the narrow funnel will usually be favored and the number of arguments that can effect the reachable goals will be reduced.

6 Applications of Funnel Theory

If the thesis of this chapter is valid, then is should be possible to explore complex spaces very simply. This section tests that thesis and describes the TAR2 *treatment learner*. TAR2 is a deliberately broken machine learner. The algorithm includes an exponential time sub-routine. If many variables are required to control a device, this sub-routine should make the whole TAR2 system impractical. However, as we shall see, TAR2 works in many domains. TAR2 is therefore both the *test* and the *application* of funnel theory. The *more* TAR2 works, the *more* we believe that many maybes mean mostly the same thing. Further, if many maybes means mostly the same thing, then a simple controller like TAR2 will be applicable in many domains.

To understand TAR2, consider the log of golf playing behavior seen in Figure 11. In that log, we only play *lots* of golf in $\frac{6}{5+3+6} = 43\%$ of cases. To improve our game, we might search for conditions that increases our golfing frequency. Two such conditions are shown in the `WHERE` test of the select statements in Figure 11. In the case of `outlook=overcast`, we play *lots* of golf all the time. In the case of `humidity` $\leq$ `90`, we only play *lots* of golf in 20% of cases. So one way to play lots of golf would be to select a vacation location where it was always overcast. While on holidays, one thing to watch for is the humidity: if it rises over 90%, then our frequent golf games are threatened.

The tests in the `WHERE` clause of the select statements in Figure 11 is a *treatment*. Classes in treatment learning get a score and the learner uses this to assess the class frequencies resulting from *applying a treatment* (i.e. using them in a `WHERE` clause). In normal mode, TAR2 does *controller learning* that finds a treatment which selects for better classes and reject worse classes By reversing the scoring function, treatment learning can also select for the worse classes and reject the better classes. This mode is called *monitor learning* since it finds the thing we should most watch for. In the golf example, *outlook = 'overcast'* was the controller and $humidity \geq 90$ was the monitor.

TAR2 automatically explores a very large space of possible treatments. TAR2's configuration file lets an analyst specify a search for the best treatment using conjunctions of size 1,2,3,4, etc. Since TAR2's search is elaborate, an analyst can automatically find the *best* and *worst* possible situation within a data set. For example, in the golf example, TAR2 explored all the attribute ranges of Figure 11 to learn that the *best* situation was *outlook = 'overcast'* and worst possible situation was $humidity \geq 90$.

TAR2 has been applied to many domains. The algorithm shouldn't work but it has proven successful in many domains (see [24, 34, 35, 36, 29]). Theoretically, TAR2 is intractable since there are an exponential number of possible attribute ranges to explore. TAR2 culls the space of possible attribute ranges using a heuristic *confidence1* measure that selects attribute ranges that are more frequent in good classes than in poorer classes. However, this heuristic is pretty dumb: it was merely the first one we could think of.

outlook	*temp(°F)*	*humidity*	*windy?*	*class*
sunny	*85*	*86*	*false*	*none*
sunny	*80*	*90*	*true*	*none*
sunny	*72*	*95*	*false*	*none*
rain	*65*	*70*	*true*	*none*
rain	*71*	*96*	*true*	*none*
rain	*70*	*96*	*false*	*some*
rain	*68*	*80*	*false*	*some*
rain	*75*	*80*	*false*	*some*
sunny	*69*	*70*	*false*	*lots*
sunny	*75*	*70*	*true*	*lots*
overcast	*83*	*88*	*false*	*lots*
overcast	*64*	*65*	*true*	*lots*
overcast	*72*	*90*	*true*	*lots*
overcast	*81*	*75*	*false*	*lots*

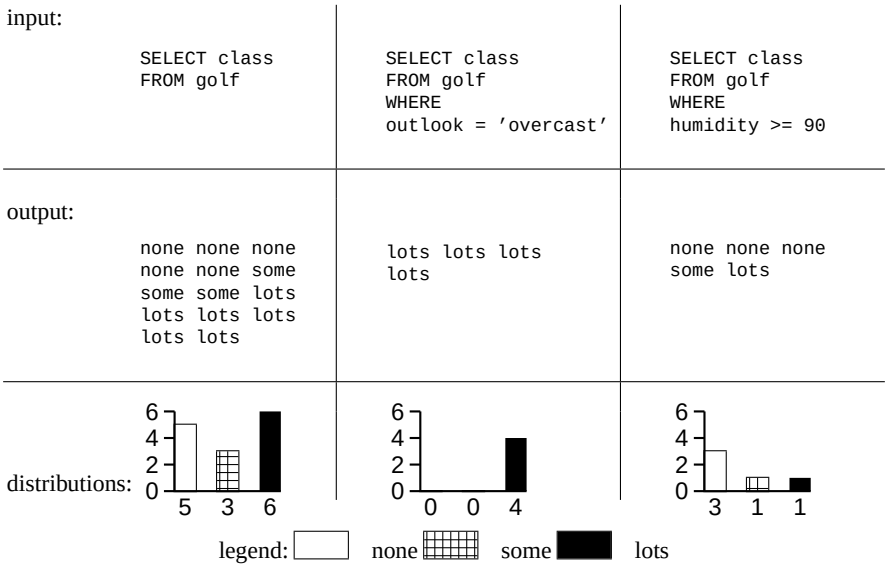

Fig. 11. Class distributions selected by different conditions.

The repeated success of TAR2 in many domains is inexplicable *unless* narrow funnels are common.

Due to the numerous examples of TAR2's success, only a sample of its applications are shown below. One way to assess TAR2 is to test how well it can control some model. To perform such an assessment, we (i) generated data sets from some model; (ii) apply TAR2 to find treatments from those data set; (iii) imposed those treatments as constraints on the model; (iv) ran the model a second time; (v) compared the outputs of the second run to the predictions made by TAR2.

In the following simulations studies, a *baseline* class distribution was used by TAR2 to generate a best controller and a prediction of how this best controller would change the class distribution. We call the predicted distribution the *treated* distribution. The *actual* distribution was the class distribution seen after the best controller was imposed on the model and the model executed again. In Figure 12 and Figure 13, the treated distribution matches the result distribution almost exactly; i.e. TAR2 accurately predicted the effects of the controller treatment.

Figure 12 was generated from a model of software project risk. This risk model was implemented as part of the COCOMO project. The goal of the COCOMO project is to build an open-source software cost estimation model [1]. Internally, the model contains a matrix of parameters that should be tuned to a particular software organization. Using COCOMO-II, the Madachy risk model can assess the risk of a software cost overrun [28]. For machine learning purposes, the goal of using the Madachy model is to find a change to a description of a software project that reduces the likelihood of a poor risk software project [38, 34]. In the experiment shown in Figure 12, the model was executed 30,000 times using randomly selected inputs. When the treatments learnt from TAR2 treatments were imposed on model inputs, and the model was executed again, all the high risk projects were removed, the percentage of medium risk projects was significantly reduced, and the percentage of low risk projects was tripled.

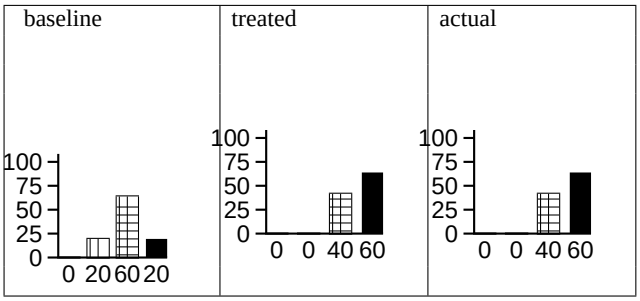

Fig. 12. COCOMO key: very high risk; high risk; medium risk; low risk.

Figure 13 shows TAR2 controlling a qualitative description of an electrical circuit. A qualitative description of a circuit of 47 wires connecting 9 light bulbs and 16 other components was coded in Prolog. The model was expressed as a set of constraints; e.g. the `sum` of the voltages of components in series is the `sum` of the voltage drop across each component. The goal of the circuit was to illuminate a space using the 9 light bulbs. The circuit is qualitative and qualitative mathematics is nondeterministic; e.g. sum of a negative and a positive value is unknown. The problem with the circuit was out-of-control nondeterminism. On backtracking, this circuit generated 35,228 different solutions to the constraints. In many of these solutions, the circuit was unacceptably

dark: only two bulbs glowing, on average (see the top histogram of Figure 13) . The goal of the machine learning was hence to find a minimal set of changes to the circuit to increase the illumination [33]. Figure 13 shows the distribution of the frequency with which bulbs glowed in a qualitative circuit description. The behavior of qualitative circuits is notoriously hard to predict [10] but TAR2 found two actions on the circuit that trebled the average number of bulbs that glowed (see the *treated* and *actual* plot of Figure 13).

Figure 14 shows a third simulation study with TAR2. Analysts at the NASA Jet Propulsion Laboratory debate satellite design by building a semantic network connecting design decisions to satellite requirements [18]. Each edge is annotated with the numeric cost and benefits of taking some action. Some of these nodes represent base decisions within the project (e.g. selection of a particular type of power supply). Each set of decisions has an associated cost. The net can be executed by selecting actions and seeing what benefits results. One such network included 90 possible actions; i.e. $2^{99} \approx 10^{30}$ combinations of actions. Note the black line, top-left, of Figure 14. All the dots below this line were generated via 10,000 random selections of the decisions, and the collection of their associated costs and benefits. All the dots above this line represent high benefit, low cost projects found by TAR2 [19]. In this application, TAR2 was used as a knowledge acquisition tool. After each run of TAR2, the proposed best controller was debated with the analysts. Each run, and its associated debate, resulted in a new set of constraints for the semantic net. The new constraints were then imposed on the model before the next run. After five runs, TAR2 found 30 decisions (out of 99) that crucially effected the cost/benefit of the satellite. Note that this means TAR2 also found 99-30=67 decisions that could be safely ignored.

For comparison purposes, a genetic algorithm (GA) was also applied to the Figure 14 domain [19]. The GA also found decisions that generated high benefit, low cost projects. However, each such GA solution commented on every possible decisions and there was no apparent way to ascertain which of these are the most critical decisions. The TAR2 solution was deemed superior to the GA solution by the domain experts, since the TAR2 solution required just 30 actions rather than the 99 demanded by the

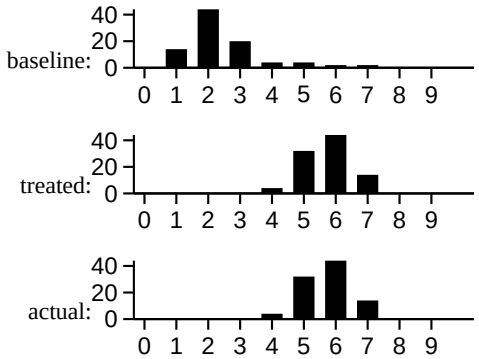

Fig. 13. Circuit. X-axis denotes number of bulbs glowing in the circuit.

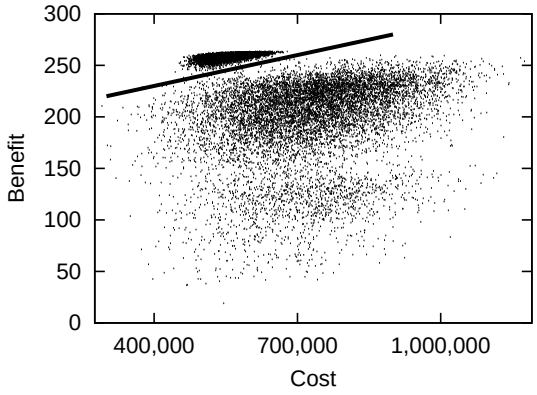

Fig. 14. Results from the satellite domain. The dots below the line show the initial output of the model: note the very large spread in the costs and benefits. The dots above the line show the final outputs of the model after 5 iterations of TAR2 learning.

GA.

Note that the Figure 14 case study is not a counter example to our thesis that most domains have narrow funnels. That study adopted the incremental approach for reasons of convenience. JPL's semantic net simulator was too slow to generate enough examples at one run. Hence, an incremental generate-and-constrain approach was taken.

7 Some Details

This section clarifies some details of this discussion.

Our case has been that *most* maybes mean the same thing, not that *all* maybes mean the same thing. As shown above in Figure 9, there exist discrete systems for which many maybes do not mean the same thing.

Also, the argument described here relates to the properties of discrete systems containing contradictions. Such an argument may not apply to continuous systems with feedback loops. Continuous systems with feedback loops can generate wildly varying behavior if that system moves into a chaotic region of its state space. Clearly, in systems experiencing such chaos, many maybes will not mostly mean the same thing.

Our emphasis on discrete systems does not preclude the application of this analysis to conventional procedural software. Much research has been devoted to the extraction of discrete models (in the form of finite state machines) from procedural code. For example, the BANDERA system [12] automatically extracts (slices) the minimum portions of a JAVA program's bytecodes which are relevant to proving particular properties models. These minimal portions are then converted into the finite state machine required for automatic formal analysis. Also, in domains where tools like BANDERA are unavailable, finite state machines can be generated from the high-level documentation describing a procedural system [45].

This chapter suggests that we can reason about a theory that contains inconsistencies. Such a suggestion might be foreign to students of classical deductive logic in which a contradiction implies anything at all. Classical deduction was a useful tool but in the late twentieth century, many researchers found that non-standard logics were required for inconsistency-tolerant reasoning about (e.g.) model-based diagnosis [11], conflicting requirements [20], or overrides in inheritance hierarchies [17].

The argument made here was that the average number of reachable goal literals are not effected greatly by the presence of contradictory inferences in a theory. This is a statement about where inference pathways *end* and not about the route taken to a goal. Hence, even when most maybes mean the same thing (i.e. the same number of goals are being reached), an indeterminate device (i.e. one containing contradictions) can take many different pathways to those goals. Consequently, the side-effects of reaching a goal can be very different. If the negation of undesirable side-effects (e.g. not reactor melt down) are added to the goal set, then the argument of this paper will apply and we can quickly check if we can/ cannot reach undesirable side effects.

This analysis assumes a set-covering semantics; i.e. we only consider literals that exist on proof trees between input and goal literals. The opposite to set-covering semantics is consistency-based semantics in which inference spreads out to find all literals consistent with inputs and goals, regardless of whether or not those literals are required for accessing some goals. The debate between set-covering and consistency-based semantics has occurred elsewhere (e.g. [26, 11]). This study favor set-covering semantics since if we are interested in literals outside the goal-finding proof trees, we can add them to our goal set.

8 Conclusion

As theory size or complexity grows, we will become less and less sure about the assertions in that theory. Contradictory options (the "maybes") will often be entered into theories, particularly if that theory is generated from designers with different views about a domain or the purpose of a program.

An often repeated experimental observation is that a fast random exploration of a program will reach as many interesting goals as a larger number of considered probes. The mathematics of reachability shows us that these observations are not some quirk of particular domains. Rather these observations are examples of a general principle: on average, the way we resolve contradictions does not effect the overall number of reachable goals *provided that* we are probing into our theories to a non-trivial depth. The TAR2 experience is that simple Monte Carlo simulations satisfy this probing depth requirements.

These mathematical and experimental results can be explained using funnel theory. Given a choice of M arguments or N arguments ($M < N$) to reach a goal, random search will usually favor the smaller set of arguments. Hence, fewer critical factors will change the number of goals we can reach, most maybes will mean the same thing, and we can use very simple methods (like TAR2) to control complex and uncertain spaces.

Acknowledgements

The arguments in this chapter benefitted greatly from numerous lively discussions with Helen Burgess and Steve Easterbrook.

References

1. C. Abts, B. Clark, S. Devnani-Chulani, E. Horowitz, R. Madachy, D. Reifer, R. Selby, and B. Steece. COCOMO II model definition manual. Technical report, Center for Software Engineering, USC,, 1998. `http://sunset.usc.edu/COCOMOII/cocomox.html#downloads`.
2. A. Acree. *On Mutations*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1980.
3. 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.
4. J. Bieman and J. Schultz. An empirical evaluation (and specification) of the all-du-paths testing criterion. *Software Engineering Journal*, 7(1):43–51, 1992.
5. D. Bobrow, S. Mittal, and M. Stefik. Expert systems: Perils and promise. *Communications of the ACM*, 29:880–894, 1986.
6. B. Buchanan, D. Barstow, R. Bechtel, J. Bennet, W. Clancey, C. Kulikowski, T. Mitchell, and D. 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.
7. B. Buchanan and E. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.
8. T. Budd. *Mutation analysis of programs test data*. PhD thesis, Yale University, 1980.
9. J. Caraca-Valente, L. Gonzalez, J. Morant, and J. Pozas. Knowledge-based systems validation: When to stop running test cases. *International Journal of Human-Computer Studies*, 2000. To appear.
10. D. Clancy and B. Kuipers. Model decomposition and simulation: A component based qualitative simulation algorithm. In *AAAI-97*, 1997.
11. L. Console and P. Torasso. A Spectrum of Definitions of Model-Based Diagnosis. *Computational Intelligence*, 7:133–141, 3 1991.
12. J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasarenu, Robby, and H. Zheng. Bandera: Extracting finite-state models from java source code. In *Proceedings ICSE2000, Limerick, Ireland*, pages 439–448, 2000.
13. R. Cordero, M. Costamagna, and E. Paschetta. A genetic algorithm approach for the calibration of cocomo-like models. In *12th COCOMO Forum*, 1997.
14. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.
15. P. Davies. Planning and expert systems. In *ECAI '94*, 1994.
16. M. Dorigo and L. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, Apr. 1997.
17. D. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *AAAI-83*, pages 104–108, 1983.
18. M. Feather, S. Cornford, and T. Larson. Combining the best attributes of qualitative and quantitative risk management tool support. In *15th IEEE International Conference on Automated Software Engineering, Grenoble, France*, pages 309–312, September 2000.

19. M. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002. Available from `http://tim.menzies.com/pdf/02re02.pdf`.
20. A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specification. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.
21. W. Gutjhar. Partition vs. random testing: The influence of uncertainty. *IEEE Transactions on Software Engineering*, 25(5):661–674, September/October 1999.
22. D. Harmon and D. King. *Expert Systems: Artificial Intelligence in Business*. John Wiley & Sons, 1983.
23. M. Harrold, J. Jones, and G. Rothermel. Empirical studies of control dependence graph size for c programs. *Empirical Software Engineering*, 3:203–211, 1998.
24. Y. Hu. Better treatment learning, 2003. Masters Thesis, Department of Electrical Engineering, University of British Columbia, in preperation.
25. Y. Iwasaki. Qualitative physics. In P. C. A. Barr and E. Feigenbaum, editors, *The Handbook of Artificial Intelligence*, volume 4, pages 323–413. Addison Wesley, 1989.
26. K. Konoligue. Abduction versus Closure in Causal Theories. *Artificial Intelligence*, 53:255–272, 1992.
27. N. Leveson. *Safeware System Safety And Computers*. Addison-Wesley, 1995.
28. R. Madachy. Heuristic risk assessment using cost factors. *IEEE Software*, 14(3):51–59, May 1997.
29. T. Menzies, E. Chiang, M. Feather, Y. Hu, and J. Kiper. Condensing uncertainty via incremental treatment learning. In *Annals of Software Engineering (submitted)*, 2002. Available from `http://tim.menzies.com/pdf/02itar2.pdf`.
30. T. Menzies, B. Cukic, H. Singh, and J. Powell. Testing nondeterminate systems. In *ISSRE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00issre.pdf`.
31. T. Menzies and B. Cukic. Adequacy of limited testing for knowledge based systems. *International Journal on Artificial Intelligence Tools (IJAIT)*, June 2000. Available from `http://tim.menzies.com/pdf/00ijait.pdf`.
32. T. Menzies, S. Easterbrook, B. Nuseibeh, and S. Waugh. An empirical investigation of multiple viewpoint reasoning in requirements engineering. In *RE '99*, 1999. Available from `http://tim.menzies.com/pdf/99re.pdf`.
33. T. Menzies and Y. Hu. Constraining discussions in requirements engineering. In *First International Workshop on Model-based Requirements Engineering*, 2001. Available from `http://tim.menzies.com/pdf/01lesstalk.pdf`.
34. T. Menzies and Y. Hu. Reusing models for requirements engineering. In *First International Workshop on Model-based Requirements Engineering*, 2001. Available from `http://tim.menzies.com/pdf/01reusere.pdf`.
35. T. Menzies and Y. Hu. Agents in a wild world. In C. Rouff, editor, *Formal Approaches to Agent-Based Systems, book chapter*, 2002. Available from `http://tim.menzies.com/pdf/01agents.pdf`.
36. T. Menzies and Y. Hu. Just enough learning (of association rules). In *WVU CSEE tech report*, 2002. Available from `http://tim.menzies.com/pdf/02tar2.pdf`.
37. T. Menzies and 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.
38. T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00ase.pdf`.

39. T. Menzies. *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales, 1995. Ph.D. thesis. Available from `http://tim.menzies.com/pdf/95thesis.pdf`.
40. T. Menzies. Critical success metrics: Evaluation at the business-level. *International Journal of Human-Computer Studies, special issue on evaluation of KE techniques*, 51(4):783–799, October 1999. Available from `http://tim.menzies.com/pdf/99csm.pdf`.
41. C. Michael. On the uniformity of error propagation in software. In *Proceedings of the 12th Annual Confererence on Computer Assurance (COMPASS '97) Gaithersburg, MD*, 1997.
42. C. L. Ramsey and V. Basili. An evaluation for expert systems for software engineering management. *IEEE Transactions on Software Engineering*, 15:747–759, 1989.
43. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *AAAI '92*, pages 440–446, 1992.
44. J. Shavlik, R. Mooney, and G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991.
45. J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE). Limerick, Ireland*, June 2000. Available from `http://www.riacs.edu/research/detail/ase/icse2000.ps.gz`.
46. W. Wong and A. Mathur. Reducing the cost of mutation testing: An empirical study. *The Journal of Systems and Software*, 31(3):185–196, December 1995.
47. V. Yu, L. Fagan, S. Wraith, W. Clancey, A. Scott, J. Hanigan, R. Blum, B. Buchanan, and S. Cohen. Antimicrobial Selection by a Computer: a Blinded Evaluation by Infectious Disease Experts. *Journal of American Medical Association*, 242:1279–1282, 1979.
48. L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-3(1):28–44, Jan. 1973.