Chapter 1

AGENTS IN A WILD WORLD

Tim Menzies, Ying Hu
Department Electrical and Computer Engineering
University of British Columbia
tim@menzies.com, huying_ca@yahoo.com

Abstract Agents try to achieve goals while being buffeted by wild factors outside of their control. In the average case, it is possible for agents to (i) learn effective and cheap control strategies, despite these wild influences; and (ii) redesign devices in order to increase their immunity to wild influences.

Keywords: Agents, nondeterminism, reachability, treatment learners.

Introduction

A Turing machine is a lonely machine. It talks to no one and no outsider talks to it. Within its walls, nothing happens unless its read-write device changes an entry on a Turing tape. No external force ever stops the tape rolling backwards and forwards, searching for its solitary conclusions.

An agent machine is a social machine. Situated within some environment, the agent busily attempts to achieve goals while being buffeted by *wild forces* outside of its control. Beliefs that seemed reasonable only an hour ago may become obsolete. It is as if the Turing tape of our agent machine can be *wildly scribbled on* by the environment, without the agent noticing[1].

To assess the impacts of wild interaction, this chapter will consider an agent dealing with devices of the form of Figure 1.1. Such devices contain *hidden* variables, *inputs* variables and *observable* variables. For the purposes of this analysis, it is irrelevant if the devices are sub-routines within the agent or sensors and actuators attached to the environment. In either case, our agent only has change the *controllable* inputs. The

$$\overbrace{\underbrace{a,b,\ldots}_{\substack{\textit{controllable}\\\textit{by agent}}}\quad\underbrace{d,e,\ldots}_{\textit{wild}}}^{\textit{inputs}}\ \overbrace{g,h,j,k,l,m,n,o,p,q,r,\ldots}^{\textit{hidden}}\ \overbrace{\underbrace{v,w,\ldots}_{\substack{\textit{assessable}\\\textit{by oracle}}}\quad\underbrace{\ldots,y,z}_{\textit{ignored}}}^{\textit{outputs}}$$

Figure 1.1. An agent with wild inputs.

other inputs are *wild* and are set by forces outside of the control of the agent; e.g. the environment, or the actions of other agents. We will assume no knowledge of the wild inputs. In this model, nondeterminism is modelled as wild variables that control choice operators at all nondeterministic choice points within the hidden variables.

We will assume that some of the observables are *assessable* by some oracle. This oracle generates a small number of *scored classes* that offer a coarse-grain assessment of the behavior of a system; e.g. "very good", "good", "fair", "poor", "very poor". For example, if the device can achieve N output goals, then the oracle could output "good" if more than (say) 80% of those goals are reached.

A *treatment* is some setting to the controllable inputs. The task of our agent is to *minimize* the *cost* of both *finding* and *applying* its treatments while *maximizing* the *score* of the classes generated by the oracle. We will assume that it *costs* some effort to adjust the control inputs. The rest of this article assumes all controls cost "1" unit of effort (and this could be simply adjusted if required).

The challenge facing our agent is that much of the device is *hidden* or *wild.* Any treatments must be effective over the range of possible behaviors that result from the *wild* inputs impacting the *hidden* variables. Hence, we say that the range of possible outputs from the device represents a space of what-if scenarios and a treatment *restricts* the space of possible what-ifs by restricting the space of settings to the control variables.

This chapter argues that:

CLAIM1: In the average case, it is possible to learn effective and cheap treatments, despite the wild variables.

CLAIM2: It is possible to redesign particular devices in order to *increase* their immunity to wild variables.

CLAIM1 will be defended using an average case analysis, some experimentation, and a literature review. Such an average case analysis

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

% facts
diet(fatty).
diet(light).

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

Figure 1.2. A theory.

may not apply to particular systems. Hence **CLAIM2** is required in order to increase the chances that a particular system exhibits our desired average case behavior.

To demonstrate these claims, we need:

1 A formal model of the hidden variables. We will assume that the hidden variables form a *negation-free horn clause theory.* We will further assume that some *nogood* predicate can report pairs of incompatible variables. A sample negation-free horn clause theory and a *nogood* predicate are shown in Figure 1.2.

2 A definition of the execution method of that theory. We will assume that execution means the construction of consistent proof trees across the horn clauses, where *consistent* is defined with respect to the *nogood* predicates.

3 A theoretical sensitivity result that shows that adequate treatments can be developed for our device *across the entire range of possible wild inputs.* We will show that the entire range of possible wild inputs will, in the average case, result in a limited number of settings to the assessable outputs.

The rest of the chapter is in three parts. §1 offers a theoretical argument for **CLAIM1**. §2 discusses empirical results that support our theory. §3 argues for **CLAIM2**. Related work is discussed throughout this chapter. This work extends earlier studies on testing nondeterministic systems (Menzies and Cukic, 2000b; Menzies et al., 2000; Menzies and Cukic, 2000a).

$$\begin{array}{l}
Path_1\ \texttt{happy} \leftarrow \texttt{tranquility(hi)} \leftarrow \texttt{conscience(clear)} \\
Path_2 : \texttt{happy} \leftarrow \texttt{tranquility(hi)} \leftarrow \texttt{satiated} \leftarrow \texttt{diet(fatty)} \\
Path_3 : \texttt{happy} \leftarrow \texttt{and1} \left\{ \begin{array}{l} \leftarrow \texttt{rich} \\ \leftarrow \texttt{healthy} \leftarrow \texttt{diet(light)} \end{array} \right.
\end{array}$$

Figure 1.3. Three pathways to `happy` generated across Figure 1.4 (which is a graphical form of Figure 1.2).

Note that this analysis will only be an *average case analysis.* Such an argument for *average case* success says little about our ability to handle *uncommon, critical cases.* Hence, our analysis must be used with care if applied to safety-critical software. On the other hand, many applications are not safety-critical since such software costs in excess of one million dollars per thousand lines of code (Schooff and Haimes, 1999, p276); i.e. safety-critical applications are cost-justified in only a minority of cases.

1. Usual Case Analysis of Wild Influences

In this section, we begin our case for **CLAIM1**; i.e. in the average case, it is possible to learn effective and cheap treatments, despite the wild variables. Wild variables inject a degree of randomness into how a device will operate. We show here that such a random search within a device containing *nogoods* can be controlled by a small number of key *funnel variables* (defined below). It will be argued that:

1 The range of behaviors within a device containing *nogoods* is small if there are few possible *worlds of belief* (defined below).

2 The total number of possible *worlds of belief* is small if the *funnel size* is small.

3 In the average case, the funnel size is indeed quite small.

These three points are necessary, but not sufficient preconditions for **CLAIM1**. The next section (§2) will show that the small number of key variables within the funnel can be controlled using just the available controllable inputs.

1.1 Worlds of Belief

A theorem prover exploring a device containing *nogood* variables often reaches incompatible options; e.g. pairs of variables depreciated by the *nogood* predicate. For example, in Figure 1.2, two such incompatible beliefs are `diet(light)` and `diet(fatty)`.

When such incompatible pairs are reached, the theorem prover must choose which variable to believe. As the reasoning progresses, this continual choice of what to believe leads to a particular *world of belief.*

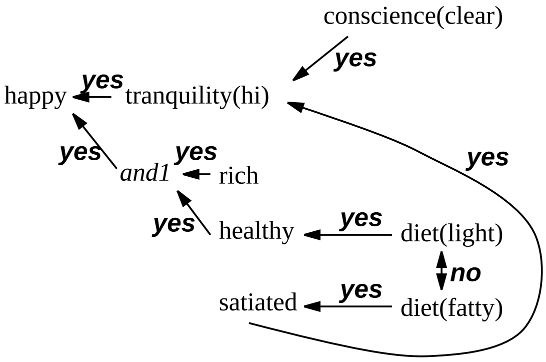


Figure 1.4. The rules of Figure 1.2 converted to an and-or graph. All nodes here are or-nodes except *and1*. All parents of an and-node must be believed if we are to believe and-node. In this graph *no-edges* represent illegal pairs of inferences; i.e. things we can't believe at the same time such as `diet(light)` and `diet(fatty)`. All other edges are *yes-edges* which represent legal inferences.

Given a model such as Figure 1.2 and a goal such as `happy`, then worlds of belief can be generated as follows.

- The individual pathways to the goals are collected. Figure 1.3 shows the possible pathways are $Path_1 ... Path_3$, shown in Figure 1.3.
- Anything that has not been asserted as a fact is an *assumption.* If we have asserted `happy`, then $Path_2$ contains the assumptions `tranquility(hi)` and `satiated` and `diet(fatty)`.
- No path can contain mutually exclusive assumptions or contradict the goal; i.e. assuming ¬`happy` is illegal since, in this example, we already believe `happy`.
- The generated pathways should be grouped together into maximal consistent subsets. Each such subset is a world of belief. Our example generates two worlds: $World_1 = \{Path_1, Path_3\}$ and $World_2 = \{Path_1, Path_2\}$.
- A world contains what we can conclude from and-or graphs. A goal is proved if it can be found in a world.

Worlds are generated when inconsistencies are detected. There are at least three methods to control such generation:

Method A- full worlds search: Method A is to generate a world for every possible resolution. Method A takes a long time[2] and pragmatic agents should avoid it.

Method B- random worlds search: Method B is to generate a world for one on resolution, picked at random, then continue on. Method B is often combined with a "rest-retry" mechanism. That is, method B is applied X times, with the system reset between each application.

Method C- use of domain knowledge: Method C is to carefully select one resolution, based on domain knowledge. This chapter is focused on the effects the unknown knowledge (the wild variables) than the effects of known knowledge; i.e. Method C is not of interest here.

Random worlds search (method B) best simulates the impact of wild inputs. Such a partial random search will find only some subset of the possible paths, particularly if it is run for a heuristically selected time interval. That is, random worlds searching may not reliably infer that (e.g.) `healthy` is an uncertain conclusion. Depending on how the conflict between `diet(light)` and `diet(happy)` is resolved at runtime, this system will sometimes nondeterministically conclude `healthy` and sometimes it won't.

CLAIM1 can be formally expressed as follows: the wild inputs do not greatly influence what worlds are explored. Traditional complexity results are pessimistic about such a claim. If a graph lacks *nogoods*, then it can be searched very fast (see the linear-time and-or graph traversal algorithm in the appendix). However, the presence of *nogoods* changes all that. Gabow et.al. (Gabow et al., 1976) showed that building pathways across programs with impossible pairs (e.g. variables marked by a *nogood* predicate) is NP-complete for all but the simplest programs (a program is very simple if it is very small, or it is a simple tree, or it has a dependency networks with out degree ≤ 1). The traditional view is that NP-complete tasks are only practical when incomplete domain-specific heuristics are applied to constrain the task. The applicable to such heuristics to all possible inputs cannot be proved. Hence, in the traditional view, using such heuristics can produce *wildly* differing results when:

I. *Not enough worlds are generated to cover the range of possible conclusions.* This first case could arise from heuristics pruning possible inferences at runtime. Random worlds search can suffer from this problem.

II. *Too many worlds are generated and we are swamped with possibilities.* This second case arises when heuristics fail to prune the search space. Full worlds search can suffer from this problem.

Both problems are removed if *the total number of possible worlds is small.* If so, then:

- Problem I disappears if all the possible conclusions can be reached by sampling just a few worlds.

- Problem II disappears if a large number of worlds is not possible.

1.2 Funnels Control Worlds

This section argues that the number of worlds is small if the *funnel* size is small. Before describing the funnel, we first categorize assumptions into three important groups: the dependent, the non-controversial, and the remaining variables that lie in the *funnel.* Only this third group of *funnel variables* will determine how many worlds are generated.

Some assumptions are *dependent* on other assumptions. For example, in $Path_2$, the `tranquility(hi)` assumption depends fully on `satiated` which, in turn, fully depends on `diet(fatty)`. In terms of exploring all the effects of different assumptions, we can ignore the dependent assumptions.

Another important category of assumptions are the assumptions that contradict no other assumptions. These *non-controversial* assumptions are never at odds with other assumptions and so do not effect the number of worlds generated. In our example, the non-controversial assumptions are everything except `diet(light)` and `diet(healthy)`. Hence, like the dependent assumptions, we will ignore these non-controversial assumptions.

The remaining assumptions are the *controversial, non-dependent* assumptions or the *funnel* assumptions. These funnel assumptions control how all the other assumptions are grouped into worlds of belief. DeKleer's key insight in the ATMS research was that a multi-world reasoning device need only focus on the funnel (DeKleer, 1986)[3]. When switching between worlds, all we need to resolve is which funnel assumptions we endorse. Continuing our example, if we endorse `diet(light)` then all the conclusions in $World_2$ follow and if we endorse `diet(healthy)` then all the conclusions in $World_1$ follow.

Paths meet and clash in the funnel. If the size of the funnel is very small, then the number of possible clashes is very small and the number of possible resolutions to those clashes is also very small. When the number of possible resolutions is very small, the number of possible worlds is very small and random search can quickly probe the different worlds of beliefs (since there are so few of them). Hence, if we can show that the average size of the funnel is small, then **CLAIM1** is supported.

1.3 Average Funnel Size

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

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

We say that the probability of reaching the goal is the value *reached.*

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? The following analysis is taken from (Menzies and Cukic, 2001) which is a simplification of (Menzies and Singh, 2001).

To find the average funnel size, we begin with the following definitions. Let the M funnel use m variables and the N funnel use n variables. For comparison purposes, we express the size of the wider funnel as a ratio α of the narrower funnel; i.e.

$$n = \alpha m \tag{1.1}$$

Each member of M is reached via a path with probability a_i while each 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 narrow pathway is

$$narrow = c\prod_{i=1}^{m} a_i \tag{1.2}$$

while the probability of reaching the goal via the wide pathway is

$$wide = d\prod_{i=1}^{n} b_i \tag{1.3}$$

Let $P(narrow|reached)$ and $P(wide|reached)$ denote the conditional probabilities of using one of the funnels, given that the goal is reached. The ratio R of these conditional probabilities informs us when the narrow funnel is favored over the wider funnel.

$$R = \frac{P(narrow|reached)}{P(wide|reached)} = \frac{\left(\frac{narrow}{reached}\right)}{\left(\frac{wide}{reached}\right)} = \frac{narrow}{wide} \tag{1.4}$$

Narrow funnels are more likely than wider funnels when

$$R \gg 1 \tag{1.5}$$

To compute the frequency of Equation 1.5, we have to make some assumptions about the probability distributions of *narrow* and *reached.* (Menzies and Singh, 2001) showed that if a_i and b_i come from uniform probability distributions, then narrow funnels are more likely than wide funnels. In the case of such uniform distributions,

$$\sum_{i=1}^{m} a_i = 1 \therefore a_i = \frac{1}{m} \therefore narrow = c\left(\frac{1}{m}\right)^m \tag{1.6}$$

Similarly, under the same assumptions,

$$wide = d\left(\frac{1}{n}\right)^n \tag{1.7}$$

Under this assumption of uniformity, $R > 1$ when

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

Recalling that $n = \alpha m$, this expression becomes

$$(\alpha m)^{\alpha m} m^{-m} > \frac{d}{c} \tag{1.8}$$

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

$$(4m)^m > \frac{d}{c} \tag{1.9}$$

At $m = 2$, Equation 1.9 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.

To explore the case where $\sum_{i=1}^{m} a_i \neq 1$ and $\sum_{i=1}^{m} b_i \neq 1$ (i.e. the non-uniform probability distribution case), we created and executed a small simulator many times. In this simulator, we found the frequency at which $R > t$ where t was some threshold value.

To execute the simulator, we required some knowledge of the distributions of *narrow* and *wide* when they are computed by nondeterministic search.Those distributions were taken from an average case analysis of reachability across graphs such as Figure 1.4. This reachability analysis is discussed below.

1.3.1 A Reachability Model. Menzies, Cukic, Singh and Powell (Menzies et al., 2000) computed the odds of reaching some random part of a space of nondeterminate choices from random inputs. The analysis assumed that software had been transformed into a possibly cyclic directed graph containing and-nodes and or-nodes; e.g. Figure 1.2 has been converted to Figure 1.4. A simplified description of their analysis is presented here. For example, in the full model, all variables are really random gamma or beta distribution variables specified by a *mean* and a *skew* parameter; see (Menzies et al., 2000) for details.

Assume that "*in*" number of inputs have been presented to a graph containing V nodes. From these inputs, we grow a tree of pathways down to some random node within the graph. The odds of reaching a node straight away from the inputs is

$$x_0 = \frac{in}{V} \tag{1.10}$$

The probability of reaching an and-node with *andp* parents is the probability of reaching all its parents; i.e.

$$x_{and} = x_i^{andp} \tag{1.11}$$

where x_i is the probability we computed in the prior step of the simulation (and x_0 being the base case). The probability of reaching an or-node with *orp* parents is the probability of not missing any of its parents; i.e.:

$$x_{or} = 1 - (1 - x_i)^{orp} \tag{1.12}$$

If the ratio of and-nodes in a network is *andf*, then the ratio of or-nodes in the same network is $1 - andf$. The odds of reaching some random node x_j is the weighted sum of the probabilities of reaching and-nodes or-nodes; i.e.

$$\begin{aligned} x_j &= andf * x_{and} + orf * x'_{or} \quad (1.13) \\ x'_{or} &= x_{or} * x_{no\,loop} * x_{no\,clash} \quad (1.14) \end{aligned}$$

X'_{or} is similar to the original x_{or}, but it is modified by Equation 1.14, for the following reasons. Recall from Figure 1.2 that some nodes are *nogood* with other nodes and the average size of the *nogood* set for each variable is *no*. The probability $x_{no\,clash}$ is that a new node can be added to the tree of pathways of size $size_j$ at level j is the probability that this new node will not contradict any of the or-nodes in the current path:

$$x_{no\,clash} = \left(1 - \frac{no}{V}\right)^{size_j * orf} \tag{1.15}$$

Not only must a new node not contradict with other nodes in the tree of pathways, it must also not introduce a loop into the tree, since loops do not contribute to revealing unseen nodes.

$$x_{no\,loop} = \left(1 - \frac{1}{V}\right)^{size_j * orf} \tag{1.16}$$

Observe the use of $size_j * orf$ in Equation 1.15 and Equation 1.16. 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.

Having calculated x_j, we can convert it to the number of tests N required to be 99% sure of find a fault with probability x_j as follows. Equation 1.12 is really the sampling-with-replacement equation where *orp* is the number of trials N. We can use sampling-with-replacement to find the certainty of finding some event after N trials. If we demand a 99% certainty of reaching a node at step j (i.e. $y = 0.99$), then we can re-arrange Equation 1.12 to

$$N(x_j) = \frac{log(1 - 0.99)}{log(1 - x_j)} \tag{1.17}$$

After 150,000 simulations of this model, some best and worst cases were identified. These are shown in Figure 1.5 labelled *pessimistic* and *optimistic* respectively. In the pessimistic case, we restricted the depth of our search to some trivial size: $j < 10$. In this pessimistic case, more than 10,000 random inputs are required to reach half the nodes in the graphs we simulated. In the optimistic case, we gave the search engine greater freedom to explore: $j < 100$. In this optimistic case, less than 100 random inputs would reach over half the nodes in the graphs we simulated.

1.3.2 Simulating Funnels. Having some knowledge of the distributions, we can now compute the frequency of Equation 1.5 (i.e.

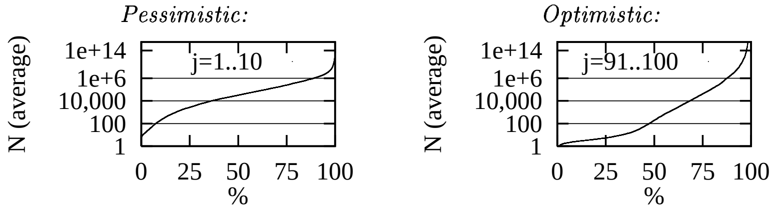


Figure 1.5. 150,000 runs of the simulator generated x_j figures which were converted into number of tests N required using Equation 1.17. X-axis shows the percentile distribution of the output of the runs.

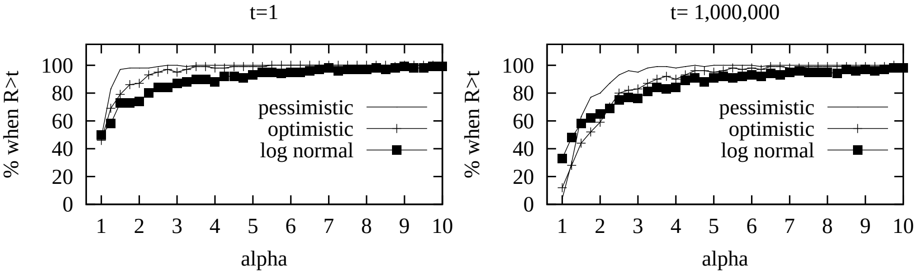


Figure 1.6. 10000 runs of the funnel simulator. Y-axis shows what percentage of the runs satisfies $R > t$.

$R \gg 1$) for non-uniform distributions. For one run of the Equation 1.4 simulator, m and α were picked at random from the ranges:

$$m \in \{1, 2, \ldots 10\}; \quad \alpha \in \{1, 1.25, 1.5, \ldots 10\}$$

The a_i, b_i, c, d needed for Equation 1.2 and Equation 1.3 were taken from one of three distributions: the pessimistic and optimistic curves shown in Figure 1.5, plus a log normal curve (just for comparison purposes). For the log-normal curve, mean μ and standard deviation σ^2 of the logarithm of the variables were picked at random from the following ranges:

$$\mu \in \{1, 2, \ldots 10\}; \quad spread \in \{0.05, 0.1, 0.2, 0.4, 0.8\}$$

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

$$\sigma^2 = spread * \mu; \quad probability = 10^{-1*normDist(\mu,\sigma^2)}$$

R was then calculated and the number of times R exceeded different values for t is shown in Figure 1.6. 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 $R > t$ is satisfied and the narrower funnel is preferred to the wider funnel. The effect is quite pronounced. For example, for all the studied distributions, if the wider funnel is 2.25 times bigger than the narrow funnel, random search will be 1,000,000 times as likely as to use the narrow funnel (see the lower graph of Figure 1.6). Interestingly, as reachability drops, the odds of using the narrow funnel increase (see the *pessimistic curves* in Figure 1.6). That is, the harder the search, the less likely the search will suffer from the problem of nondeterminate search under-sampling the space.

1.4 Exploiting the Funnel via Random Search

Prior research is pessimistic about finding the funnels in tractable time. Assumption-based truth maintenance systems incrementally build and update the *minimal environments* (a.k.a. funnels) that control the total assumption space (DeKleer, 1986). In practice, finding the minimal environments takes time exponential on theory size (Menzies and Compton, 1997).

However, such pessimism may be misplaced. There is no need to *find* the funnel in order to *exploit* it since *any* pathway from inputs to goals must pass through the funnel (by definition). Repeated application of some random search technique will stumble across the funnel variables, providing that search technique reaches the goals. Further, assuming small funnels, such a random search would not have to run for very long to sample all the possible worlds.

There is some evidence in the literature for this optimistic view that random search can quickly sample the behavior of a wild system:

- For CNF representations, it is well established that random search with retries can demonstrate satisfiability in theories too large for full search (Kautz and Selman, 1996).

- Williams and Nayak found that a random worlds search algorithm performed as well as the best available assumption-based truth maintenance system (which conducts a full worlds search) (Williams and Nayak, 1996).

- Menzies, Easterbrook et.al. report experiments comparing random world search with full world search for requirements engineering. After millions of runs, they concluded that randomized world search found almost as many goals in less time as full worlds search (Menzies et al., 1999).

- In other work, Menzies and Michael compared a full worlds search with a random worlds search. As expected, the full worlds search ran slow ($O(2^N)$) while the random worlds search ran much faster ($O(N^2)$); see Figure 1.7. What is more interesting is that, for problems where both search methods terminated, the random worlds

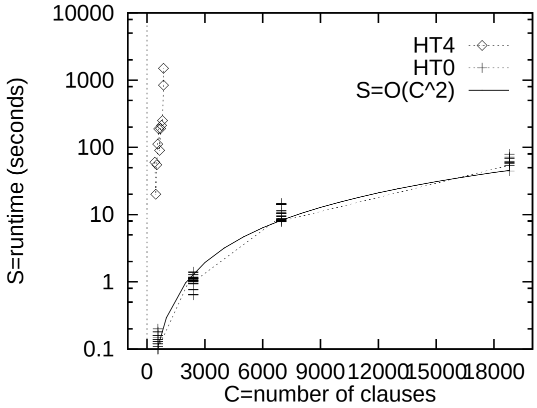


Figure 1.7. HT4: full worlds search- fork one world for each consistent possibility reached from inputs. HT0: random worlds search- when contradictions are found, pick one resolution at random, then continue. In the zone where both algorithms terminated, HT0's random world search found 98% of the goals found by HT4's full worlds search (Menzies and Michael, 1999).

search found 98% of the goals found by the full worlds search (Menzies and Michael, 1999).

2. Controlling the Funnel

The previous section argued that (i) funnel variables control a device; (ii) the number of funnel variables is small; and (iii) this small funnel can be found quickly via randomized search. This is not enough to prove **CLAIM1**. It remains to be shown that the funnel variables can be controlled using just the available controllable inputs. This section uses experiments with *treatment learners* to argue that the funnel can be influenced by the controllers.

2.1 Treatment Learners: An Overview

Treatment learners input a sample of a device's behavior that has been classified by the oracle in Figure 1.1. These learners output a conjunction that is a constraint on future control inputs of the device. This conjunction, called the *treatment*, is a control strategy that "nudges" the device away from "bad" behavior towards "good" behavior (where "good" and "bad" is defined by the oracle).

The learnt treatments must be carefully assessed. Ideally, the treatments can be applied to the device that generated the data. However, commonly, this is not practical and the learnt treatments should be studied using an N-way cross-validation study on the training data (i.e., N

times, learn on $\frac{N-1}{N}$ of the data then test on the remaining $\frac{1}{N}$-th of the data).

As we shall see, treatment learners are very simple and can only work if there exist a small number of funnel attributes that control the behavior of the systems. **CLAIM1** holds in domains where treatment learners can return adequate treatments.

The TAR2 (Menzies and Kiper, 2001; Hu, 2002) treatment learner used in these experiments is an optimization and generalization of the TAR1 system (Menzies and Sinsel, 2000). The following experiments use examples presented in this text, data generated by (Menzies and Kiper, 2001), and examples from the UC Irvine machine learning database (`http://www.ics.uci.edu/~mlearn/`. In keeping with the abstract model of Figure 1.1, examples were selected with many uncertain variables. For example, the CAR data was generated from some model of car assessment, the details of which are unavailable to us. Also, the CMM2 and REACH examples were generated from models where key control values were picked at random for each different run. In all these cases, treatment learners were able to build adequate controllers.

The following brief sketch of TAR2 skips over numerous details. For example, TAR2 generate treatments in increasing size order. That is, smaller and cheaper treatments are generated before larger and more expensive treatments. Also, when not all attributes in the input data set are controllable, TAR2 has a facility for focusing the treatments only on the controllable inputs. Further, sometimes treatments can too restrictive. This can happen when the proposed treatment contains many conjunctions and little of the input data falls within that restriction. In such a case, the treatment must be relaxed, lest TAR2 over-fits on the theory. For more on these and other details, see (Hu, 2002).

2.2 Treatment Learning: The Details

TAR2 input classified examples like those in Figure 1.8 and output a *treatment*; i.e. a conjunction of control attribute values. To find the treatments, TAR2 accesses a *score* for each classification. For a golfer, the classes in Figure 1.8 could be scored as *none=0* (i.e. worst), *some=1*, *lots=2* (i.e. best). TAR2 then seeks attribute ranges that occur more frequently in the highly scored classes than in the lower scored classes. Let $a = r$ be some attribute range (e.g. *outlook=overcast*) and $X(a = r)$ be the number of occurrences of that attribute range in class X (e.g. *lots(outlook=overcast)=4*). If *best* is the highest scoring class (e.g. $best = lots$) and *others* are the non-best class (e.g. $others = \{none, some\}$), then $\Delta_{a=r}$ is a measure of the worth of $a = r$ to improve the frequency of the *best* class. $\Delta_{a=r}$ is calculated as follows:

#	*outlook*	*temperature (°F)*	*humidity*	*windy?*	*time on course*
1	*sunny*	*85*	*86*	*false*	*none*
2	*sunny*	*80*	*90*	*true*	*none*
3	*sunny*	*72*	*95*	*false*	*none*
4	*rain*	*65*	*70*	*true*	*none*
5	*rain*	*71*	*96*	*true*	*none*
6	*rain*	*70*	*96*	*false*	*some*
7	*rain*	*68*	*80*	*false*	*some*
8	*rain*	*75*	*80*	*false*	*some*
9	*sunny*	*69*	*70*	*false*	*lots*
10	*sunny*	*75*	*70*	*true*	*lots*
11	*overcast*	*83*	*88*	*false*	*lots*
12	*overcast*	*64*	*65*	*true*	*lots*
13	*overcast*	*72*	*90*	*true*	*lots*
14	*overcast*	*81*	*75*	*false*	*lots*

Figure 1.8. A log of some golf-playing behavior.

$$\Delta_{a=r} = \sum_{X \in others} \left(\frac{(score(best) - score(x)) * (best(a=r) - X(a=r))}{|examples|} \right)$$

where $|examples|$ is the number of categorized examples; e.g. Figure 1.8 has $|examples| = 14$ entries.

The attribute ranges in our golf example generate the Δ histogram shown in Figure 1.9.i. A *treatment* is a subset of the attribute ranges with an *outstanding* $\Delta_{a=r}$ value. For our golf example, such attributes can be seen in Figure 1.9.i: they are the outliers with outstandingly large Δs on the right-hand-side.

To *apply* a treatment, TAR2 rejects all example entries that contradict the conjunction of the attribute rages in the treatment. The ratio of classes in the remaining examples is compared to the ratio of classes in the original example set. For example Figure 1.10 shows the frequency of classes in the untreated classes and after two different treatments. The *best treatment* is the one that most increases the relative percentage of preferred classes. In Figure 1.10, the best treatment is *outlook=overcast*; i.e. if we bribe disc jockeys to always forecast overcast weather, then in 100% of cases, we should be playing lots of golf.

2.3 On the Generality of Treatment Learners

Figure 1.9 shows the Δ distribution generated by TAR2 in several domains. Figure 1.9 lends support to **CLAIM1**: in nearly all the distributions there exist outstanding outliers for Δ (denoted by a black bar in Figure 1.9). The *worst* Δ distribution we have seen is the CAR distribution of Figure 1.9.ii where the maximum Δ seen is -7. But even

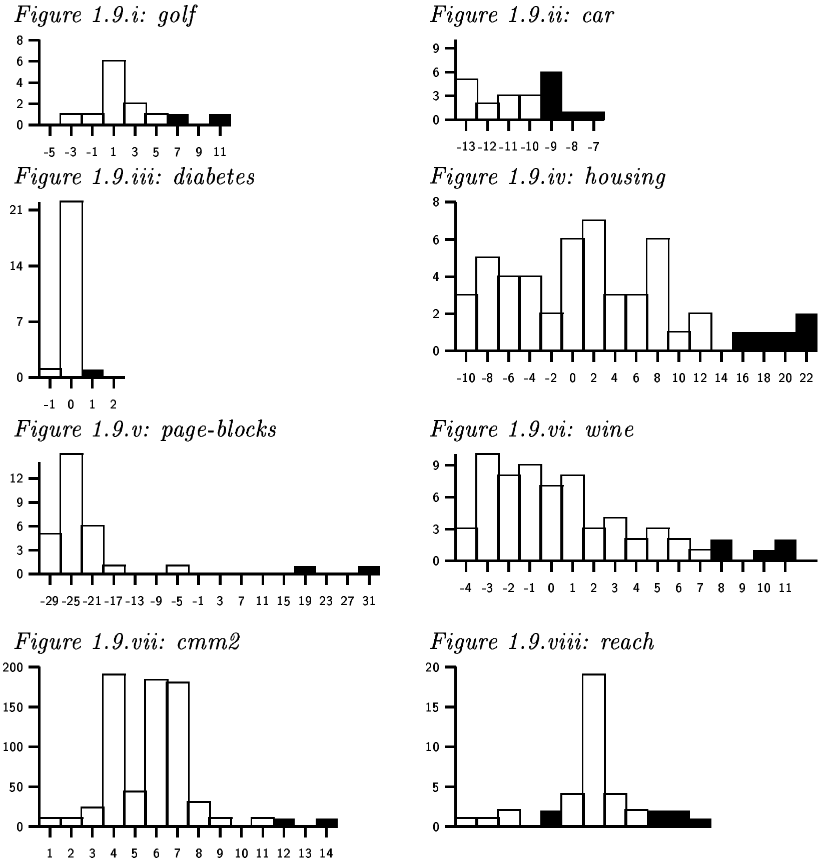


Figure 1.9.i: golf

Figure 1.9.ii: car

Figure 1.9.iii: diabetes

Figure 1.9.iv: housing

Figure 1.9.v: page-blocks

Figure 1.9.vi: wine

Figure 1.9.vii: cmm2

Figure 1.9.viii: reach

Figure 1.9. Δ distribution seen in eight data sets. Outstandingly high *deltaf* values shown in black. Y-axis is the number of times a particular Δ was seen. Figures ii,iii,iv,v,vi come from datasets taken from the UC Irvine machine learning database at `http://www.ics.uci.edu/~mlearn/`. Figures i,ii,viii are discussed in this text. Figure vii comes from data generated in (Menzies and Kiper, 2001).

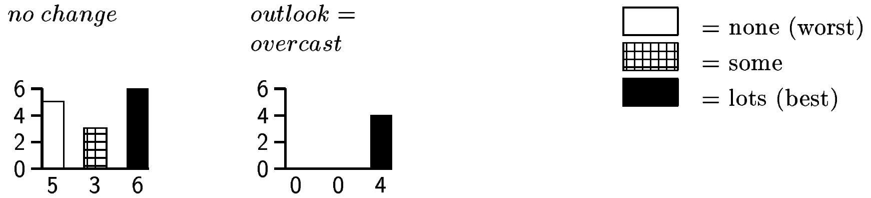


Figure 1.10. Finding treatments that can improve golf playing behavior. With no treatments, we only play golf lots of times in $\frac{6}{5+3+6} = 57\%$ of cases. With the restriction that *outlook=overcast*, then we play golf lots of times in 100% of cases.

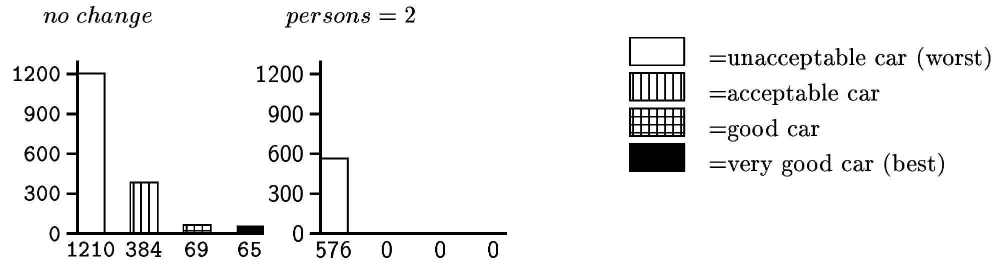


Figure 1.11. Finding treatments that can degrade cars. By reversing its scoring function, TAR2 can be made to output treatments that favor *bad* classifications. For example, in the CAR dataset from the UC Irvine machine learning database (`http://www.ics.uci.edu/~mlearn/`), $\frac{1210}{384+69+65+1210} = 70\%$ of cars are unacceptable. However, with the restriction of *person=2*, then 100% of all cars are unacceptable.

in this worst case, something like **CLAIM1** still holds. By flipping the scoring measure used on the classes, we can coerce TAR2 into finding treatments that drive the system into a worse situation; see Figure 1.11. So even when we can't advise our agents how to improve a system, we can still advise what *not* to do. In the case of CAR domain, this advise would be "don't buy two seater cars".

There is much evidence that many domains contain a small number of variables that are crucial in determining the behavior of the whole system. This phenomenon has been called various names including *small funnels* (as above), *master-variables* in scheduling (Crawford and Baker, 1994); *prime-implicants* in model-based diagnosis (Rymon, 1994); *backbones* in satisfiability (Singer et al., 2000); *dominance filtering* in design (Josephson et al., 1998); and *minimal environments* in the ATMS (DeKleer, 1986). These experimental results suggest that treatment learning in particular and **CLAIM1** in general should be widely applicable.

Note that if the outstanding Δs are all from wild variables, then TAR2 will fail. We have not seen this case in practice. In all the domains studied so far, a subset of the total domain variables were used as inputs. Usually, these variables were selected on some idiosyncratic or pragmatic grounds- e.g. these were variables for which data had already been collected. Despite idiosyncratic nature of their collection method, these attributes were sufficient to learn effective treatments.

3. Sensitivity

Optimistic conclusions derived from an average case analysis (such as shown above) may not apply to particular systems. Hence, this chapter now turns to **CLAIM2**; i.e. it is possible to redesign particular devices in order to *increase* their immunity to wild variables.

To show **CLAIM2**, we need to first describe the use of TAR2 for sensitivity analysis. As we shall see, this kind of analysis can be used to find design options that *increase* or *decrease* the likelihood that **CLAIM1** will holds for a particular device.

Suppose a designer has access several alternate versions of some device. The alternate versions may come from a range of possible design choices. Suppose further that these devices have been implemented, executed, and their output classified by some oracle. If TAR2 is applied to this output, it will generate numerous treatments, sorted by their impact on the class distribution. Our designer can study this range of treatments to answer three interesting questions:

Q1: What is the *best* variation? This best variation would be scored highest by TAR2; e.g. see Figure 1.10.

Q2: What is the *worst* variation? This worst variation can be found using the techniques described above around Figure 1.11.

Q3: How *important* is variation X? Between the *best* and *worst* treatment are a range of alternative treatments. To assess the significance of some design option, the analyst only needs to see where this option appears within this range.

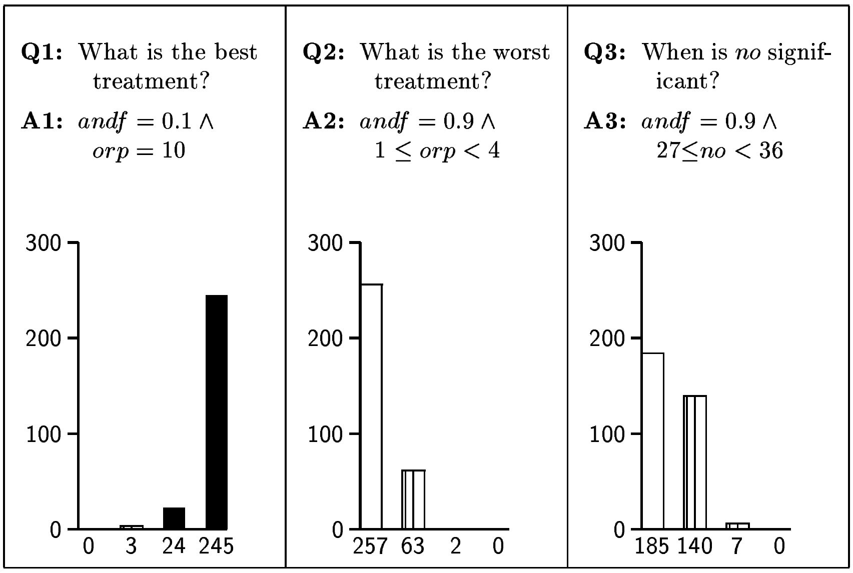
To determine what features of a device influence reachability, we only need to apply TAR2 to the 150,000 outputs collected above from Equation 1.13, classified according the scheme of Figure 1.12. The rationale for this classification scheme is as follows:

- Wild variables increase the variability in the behavior of a device.
- One measure of such variability is *reachability*; i.e. how many *randomly selected inputs probes* are required to reach *all the possible behaviors* of a system.
- We say that the *reachability is high* if the number of such probes is *small.* If the reachability is (e.g.) *impossibly* high, then an agent must conduct an impractically large number of experiments on a device (more than 10^6) in order to design control strategies.

Using Figure 1.12, TAR2 generated Figure 1.13. Figure 1.13.left can be read as follows:

$$Class = \begin{cases} \begin{array}{l} fast\, and \\ cheap \end{array} & if\ N(x_j) < 10^2, \\ \begin{array}{l} fast\, and \\ moderately \\ expensive \end{array} & if\ N(x_j) < 10^4, \\ \begin{array}{l} slow,\, and \\ expensive \end{array} & if\ N(x_j) < 10^6, \\ impossible & otherwise. \end{cases}$$

Figure 1.12. A classification scheme for output from Equation 1.13. $N(x_j)$ is the number of random experiments required to be 99% sure of reaching some randomly selected hidden variable; i.e. Equation 1.17.



Q1: What is the best treatment?

A1: *andf* $= 0.1 \wedge$ $orp = 10$

Q2: What is the worst treatment?

A2: *andf* $= 0.9 \wedge$ $1 \leq orp < 4$

Q3: When is *no* significant?

A3: *andf* $= 0.9 \wedge$ $27 \leq no < 36$

Figure 1.13. Studying the impacts of design options on reachability. Key: □ = impossible (worst); ▥ = slowCostly; ▦ = slowCheap; ■ = fastCheap (best).

If: The and-nodes are not common (*andf* = 10% of all nodes) and if or-nodes have lots of parents (*orp* = 10)...

Then: In $\frac{245}{245+24+3} = 90\%$ of cases, the reachability will be *fastCheap* (as defined in Figure 1.12).

Figure 1.13.middle can be read as follows:

If: The and-nodes are very common (*andf* = 90% of all nodes) and the or-nodes don't have many parents ($1 \leq orp < 4$)...

Then: In $\frac{257}{257+63+2} = 80\%$ of cases, the reachability will be *impossible* (as defined in Figure 1.12).

These two conclusions are somewhat trite in the sense that or-nodes relax constraints and-nodes add constraints. Naturally, if the constraints are few (as in Figure 1.13.left) then most of the device will be reachable. Similarly, if the constraints are few (as in Figure 1.13.middle), then few parts of the device will be reachable. Nevertheless these two conclusions do support **CLAIM2** since, using TAR2, it is possible to assess features of a device according to how those features change our ability to quickly understand that device.

More interesting than Figure 1.13.left and Figure 1.13.middle is Figure 1.13.right which explores the impact of nondeterminacy on reachability. Nondeterministic devices contain *nogood*s that define sets of nodes which can't be believed at the same time. The *no* variable is the average number of *nogoods* associated with a node. If $no = 0$, then the same search can take linear time using (e.g.) the `walk` algorithm shown in the appendix. If $no > 0$ then the graph contains incompatible pairs and the results of (Gabow et al., 1976) discussed near the end of §1.1 apply; i.e. a search can take exponential time to execute, in the worse case. Such an exponential time search would imply a very poor reachability. In order to avoid such worst case behavior, many researchers to reject nondeterminism. For example, Nancy Leveson comments that "nondeterminacy is the enemy of reliability" (Leveson, 1995).

Contrary to the pessimism of (e.g.) Leveson, Figure 1.13.right suggests that even when $no > 0$, then only certain ranges of *no* result in reachability problems. Figure 1.13.right was the *worst* effect ever seen in the TAR2 treatments that mentioned the *no* variable. Clearly, some ranges of *no* have a disastrous effect on reachability. In the case of $27 \leq no \leq 36$ and frequent and-nodes (*andf* = 90% of all nodes), then in $\frac{185}{185+140+7} = 56\%$ of cases, the reachability will be *impossible* (as defined in Figure 1.12). However, while *no* can determine reachability, other attributes can dominate the *no* effect. Recall that Figure 1.13.middle showed treatments that have a *worse* impact than in Figure 1.13.right *without* mentioning the *no* variable.

This kind of sensitivity analysis can be finely tuned to a particular device:

1 The parameters required for the reachability model are set using an analysis of the code in that particular project.

2 When uncertainty exists over those parameters, plausible minimum and maximum bounds for those parameters are determined.

3 Equation 1.13 is executed using random inputs picked from the preceding two points.

4 TAR2 learns treatment on the data collected from the preceding run.

4. Discussion

Traditional formal analysis makes a *deterministic assumption* where all computation is controlled. However, agents working in dynamic environments cannot access all knowledge in that domain. Nor can they stop the environment, or other agents, changing beliefs without informing our agent. Hence, this deterministic assumption may not be valid for agents working in highly dynamic environments.

We have presented here a non-standard formal analysis that makes a *nondeterministic assumption.* At issue was our ability to control a device, despite random perturbations to that device from the environment (or from other agents). Assuming that the device is of the form of Figure 1.2 (i.e. negation-free horn clauses plus some *nogood* predicates), then there will exist a *funnel* within the theory. This funnel contains the key variables that control the rest of the system. Theoretically, is has been shown that, on average, the size of the funnel is small. Hence, only a small number of different behaviors are possible, despite the inputs from the wild inputs. Also, using some experiments with the TAR2 treatment learner, it was argued that these funnel variables can be controlled from the inputs.

All the above was an average case analysis. By applying TAR2 to our theoretical model, it is possible to theoretically assess alternate designs according to how well they "nudge" a system into this desired average case behavior. Several "nudges" were discussed above include the effects of nondeterminism within a system. Compared to other factors, nondeterminism was not the most significant problem associated with search a device.

Our current research direction is to test our theory on real models reduced to our negation-free horn clauses. To this end, we are building translators from SCR state charts (Heitmeyer et al., 1996) to our and-or graphs. Once built, we will test if:

- The range of actually behaviors within these state charts are as small as predicted by funnel theory.
- The theoretical assessments of alternate designs seen in §3 will let us find new designs that are less influenced by the wild inputs.

Notes

[1] We owe this image to the keynote address of Dr. James Handler at the first NASA Goddard Workshop on Formal Approaches to Agent-Based Systems, April 5-7, 1999.

[2] Method A is a synonym for "explore every option in the program". Experience suggests that this is often an intractable process; e.g. the state-space explosion problem of model checkers (Holzmann, 1997); or the intractable branching problem of qualitative reasoning (Clancy and Kuipers, 1997).

[3] DeKleer called the funnel assumptions the *minimal environments*. We do not adopt that terminology here since DeKleer used consistency-based abduction while we are exploring set-covering abduction here. For an excellent discussion that defines and distinguishes set-covering from consistency-based methods, see (Console and Torasso, 1991).

References

Clancy, D. and Kuipers, B. (1997). Model decomposition and simulation: A component based qualitative simulation algorithm. In *AAAI-97*.

Console, L. and Torasso, P. (1991). A Spectrum of Definitions of Model-Based Diagnosis. *Computational Intelligence*, 7:133–141.

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

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

Gabow, H., Maheshwari, S., and Osterweil, L. (1976). On two problems in the generation of program test paths. *IEEE Trans. Software Engrg*, SE-2:227–231.

Heitmeyer, C. L., Jeffords, R. D., and Labaw, B. G. (1996). Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261. Available from `http://citeseer.nj.nec.com/heitmeyer96automated.html`.

Holzmann, G. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295.

Hu, Y. (2002). Treatment learning. Masters thesis, Unviersity of British Columbia, Department of Electrical and Computer Engineering. In preperation.

Josephson, J., Chandrasekaran, B., Carroll, M., Iyer, N., Wasacz, B., and Rizzoni, G. (1998). Exploration of large design spaces: an architecture and preliminary results. In *AAAI '98*. Available from `http://www.cis.ohio-state.edu/~jj/Explore.ps`.

Kautz, H. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park. AAAI Press / MIT Press. Available from `http://www.cc.gatech.edu/~jimmyd/summaries/kautz1996.ps`.

Leveson, N. (1995). *Safeware System Safety And Computers.* Addison-Wesley.

Menzies, T. and Compton, P. (1997). Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine*, 10:145–175. Available from `http://tim.menzies.com/pdf/96aim.pdf`.

Menzies, T. and Cukic, B. (2000a). Adequacy of limited testing for knowledge based systems. *International Journal on Artificial Intelligence Tools (IJAIT).* To appear. Available from `http://tim.menzies.com/pdf/00ijait.pdf`.

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

Menzies, T. and Cukic, B. (2001). Average case coverage for validation of ai systems. In *AAAI Stanford Spring Symposium on Model-based Validation of AI Systems.* Available from `http://tim.menzies.com/pdf/00validint.pdf`.

Menzies, T., Cukic, B., Singh, H., and Powell, J. (2000). Testing nondeterminate systems. In *ISSRE 2000.* Available from `http://tim.menzies.com/pdf/00issre.pdf`.

Menzies, T., Easterbrook, S., Nuseibeh, B., and Waugh, S. (1999). An empirical investigation of multiple viewpoint reasoning in requirements engineering. In *RE '99.* Available from `http://tim.menzies.com/pdf/99re.pdf`.

Menzies, T. and Kiper, J. (2001). Learning better software processes. In *Submitted to ASE-2001.* Available from `http://tim.menzies.com/pdf/01ml4re.pdf`.

Menzies, T. and Michael, C. (1999). 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`.*

Menzies, T. and Singh, H. (2001). Many maybes mean (mostly) the same thing. In *2nd International Workshop on Soft Computing applied to Software Engineering (Netherlands), February.* Available from `http://tim.menzies.com/pdf/00maybe.pdf`.

Menzies, T. and Sinsel, E. (2000). Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000.* Available from `http://tim.menzies.com/pdf/00ase.pdf`.

Rymon, R. (1994). An se-tree-based prime implicant generation algorithm. In *Annals of Math. and A.I., special issue on Model-Based Diagnosis*, volume 11. Available from `http://citeseer.nj.nec.com/193704.html`.

Schooff, R. and Haimes, Y. (1999). Dynamic multistage software estimation. *IEEE Transactions of Systems, Man, and Cybernetics*, 29(2):272–284.

Singer, J., Gent, I. P., and Smaill, A. (2000). Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270.

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

Appendix: Walking And-Or Graphs in Linear Time

The function `walk` shown below returns a list of nodes reachable across some and-or graph `g` from supplied inputs `ins`. `Walk` assumes that the graph does not contain *nogoods*.

```
STRUCT node {
    type     : one OF {and,or}
    parents  : list OF node
    kids     : list OF node
    waiting  : integer
}

FUNCTION walk(g: list of node, ins: list of node): list OF node {
    LOCAL v,k: node,
          out,todo: list OF node;
    FOR v ∈ g {v.waiting ← {IF v.type=="and" THEN |v.parents| ELSE 1 }}
    out ← ins
    todo ← ins
    WHILE (v ← pop(todo)) {
        out ← push(out,v)
        FOR k ∈ v.kids {
            k.waiting--
            IF k.waiting==0 THEN {todo ← push(todo,k)}
        }
    }
    RETURN out
}
```

On each visit to a node `k`, a `waiting` counter is decremented. When that counter goes to zero, then the algorithm has visited enough parents `v` to permit a visit to the child.. That child node is then added to a `todo` stack so that its children can be explored at subsequent iteration.

Note that *all* the parents of an and-node must be visited before visiting the and-node while only *one* parent of and or-node need be visited before we can visit and or-node. To implement this, and-nodes have their `waiting` initialized to the number of parents while or-nodes have their `waiting` set to one.

Each node `k` is visited, at maximum, once for each parent of that node `v`. Hence, the algorithm's time complexity is $O(N)$ where N is the number of nodes in the graph.