To appear: SEKE '99, June 17-19, Kaiserslautern, Germany. Available from research.ivv.nasa.gov/docs/techreports/1999/NASA-IVV-99-007.pdf. WP:b/99/seke/words(May 11, 1999).

Fewer Slices of PIE: Optimising Mutation Testing via Abduction

Tim Menzies[‡], Christoph C. Michael[†]

[‡]NASA/WVU Software Research Lab, 100 University Drive, Fairmont WV 26554 <tim@menzies.com>

[†]RST Corporation, Suite#250, 21515 Ridgetop Circle, Sterling VA 20166 <ccmich@rstcorp.com>

Abstract

A repeated observation is that test probes rapidly saturate *the search space within software: i.e. many probes of software yield only slightly more information than a few probes. HT0 is a software test engine that assumes rapid saturation. Instead of exploring all probes, randomly different probes are explored one at a time. Probe exploration halts when randomly generated probe J yields little more information than randomly generated probes $1, 2, ..I$, ($I < J$). The runtimes of HT0 were observed to be $O(N^2)$, at least for the theories studies here.*

1 Introduction

Consider a pie that may contain interesting nuggets of information. How should we slice the pie in order to find those nuggets? If the pie is software, then one method for "slicing" is the *mutation assessment* performed by the PIE system [17, 21, 22] and others [1, 4, 24]. In the *propagation analysis* of PIE, random inputs are selected from some from known distributions D (D represents known values from the domain of the program). When a program P runs using this input, certain data values are set. Conceptually, some of these values at location l are perturbed and the program is executed again (though, if the mutator has low-level access to the program P, then the perturbation can be interleaved with the original execution run). The difference in the behaviour between the initial run and the runs with perturbed values is a measure of the sensitivity of the program to errors around l. In practice, mutation analysis techniques like PIE have found faults in many systems including a medical imaging system, a train control system, a nuclear power plant, and the lethal errors that killed the Ariane 5 space rocket as well as human users of the Therac-25 radiotherapy unit [21, Cpt.7].

If the pie is big and our knife is very thin or very short, then it may take a while before we find useful nuggets of information. As with baking, so to with software. The computational cost of performing software mutation assessment is one of the major restrictions to its use [24]. For example, a sample PIE-based study would require 200,000,000 executions of the program (a number too large to be practical) [21, p126].

This article proposes a optimization to mutation testing based on the following curious observation. When cutting up pies, it makes little sense to slice in the same place twice. Strangely, it is hard to find a fresh place to slice a program. A repeated observation is that probes into software *rapidly saturate*; i.e. after a small number of probes, no new information is gained. For example, despite numerous perturbations on data values using PIE, Michael found that in 80 to 90% of cases, there were no changes in the behaviour of a range of programs [17]. Another study compared results using X% of a library of mutators, randomly selected (X $\in \{10,15,\ldots 40,100\}$). Most of what could be learnt from the program could be learnt using only X=10% of the mutators; i.e. after a very small number of mutators, new mutators acted in the same manner as previously used mutators [24]. The same observation has been made elsewhere [1, 4].

This article explores HT0, an optimization of mutation-based testing. HT0 terminates when it detects saturation; i.e. when new probes tells us little more than old probes. Experience with HT0 suggests that (e.g.) the 200,000,000 executions required above by PIE are excessive. That is, a saturation-aware mutation analysis should terminate much faster.

The rest of this article is structured as follows. HT0 is introduced in the next section and is mapped into PIE. Instead of exploring all probes, HT0 performs randomly different probes, one at a time. Probe exploration halts when randomly generated probe J yields little more information than randomly generated probes $1, 2, ..I$, ($I < J$). Experimental results follow. HT0 terminated very rapdily ($I < 6$) and runs very fast ($O(N^2)$)- at least for the theories studied here. A literature review shows that rapid saturation has been observed often in the SE/KE literature. That is, HT0 is widely-applicable since rapid saturation is a common phenomena.

```
Pr.1  foreignSales=up, ?companyProfits=up!, ?corporateSpending=up!, investorConfidence=up.
Pr.2  domesticSales=down, ?companyProfits=down!, ?corporateSpending=down!, wageRestraint=up.
Pr.3  domesticSales=down, ?companyProfits=down!, inflation=down.
Pr.4  domesticSales=down, ?companyProfits=down!, ?inflation=down!, wagesRestraint=up.
Pr.5  foreignSales=up, ?publicConfidence=up, inflation=down.
Pr.6  foreignSales=up, ?publicConfidence=up, inflation=down, wageRestraint=up.
```

Figure 1. Connections inputs to outputs: ?X=assumption; ?X!=controversial assumptions.

2 HT0

HT0 is an adaption of HT4, an *abductive validation* device. While HT4 performs an exhaustive search, HT0 is saturation-aware. Before describing HT0, we must first describe abduction, HT4, abductive validation, and its connection to PIE. For a more general discussion on abduction and its applications, see [10, 13]).

Informally, abduction is inference to the best explanation. More precisely, abduction makes whatever assumptions A that are required to reach goals G across a theory T from some inputs Ins. G may contain desired goals we wish to reach (G^+), or error cases we which to avoid (G^-). Either way, the role of a test engine is to attempt to reach as many members of G as possible.

In the general case, only some subset of the theory T', inputs Ins', and goals G' are usable ($T' \subseteq T, Ins' \subseteq Ins, G' \subseteq G$). Abduction searches for consistent assumptions which are relevant to reaching the goals; i.e.

$$T' \cup A' \cup Ins' \cup \not\vdash \bot \quad (1)$$

$$T' \cup A' \cup Ins' \vdash G' \quad (2)$$

If contradictory assumptions are generated, these are maintained in separate worlds of belief $W_1, W_2, ...$ Each world holds the subsets of T' that can be consistently believed, given the inputs and goals. HT4's abductive validation procedure searches for the world(s) with the largest intersection with G. That is, HT4 searches for the assumptions that lead to greatest coverage of G.

As an example of abductive validation, consider the hypothetical economics theory in Figure 2 (left). In the language of this theory, all theory variables have three states: `up`, `down` or `steady`. These values model the sign of the first derivative of these variables (i.e. the rate of change in each value). `X` $\stackrel{++}{\rightarrow}$ `Y` denotes that `Y` being `up` or `down` could be explained by `X` being `up` or `down` respectively. Also, `X` $\stackrel{--}{\rightarrow}$ `Y` denotes that `Y` being `up` or `down` could be explained by `X` being `down` or `up` respectively. Our theory can generate inconsistencies. Consider the case where the inputs Ins are (`foreignSales=up, domesticSales=down`) and the goals G are (`investorConfidence=up, inflation=down, wageRestraint=up`). We can infer the following inconsistency: `companyProfits=down` and `companyProfits=up`. Classical deduction would declare then that all variable assignments follow. Abduction is smarter: it searches for consistent subsets of the globally inconsistent theory. HT4 does this as follows. There are six proofs that connect Ins to G (see Figure 1). These proofs contain assumptions (variable assignments not found in $Ins \cup G$). Some of these proofs make contradictory assumptions A; e.g. `corporateSpending=up` in `Pr.1` and `corporateSpending=down` in `Pr.2`. That is, we cannot believe in `Pr.1` and `Pr.2` at the same time. If we sort these proofs into the subsets which we can believe at one time, we get worlds W_1 (Figure 2, middle) and W_2 (Figure 2, right) (in Figure 2 middle and right, ellipses are inputs, squares are outputs, and all other nodes are assumptions). W_1 is a maximal consistent subset of pathways that can be believed at the same time; i.e. (`Pr.1, Pr.5, Pr.6`). W_2 is another maximal consistent subset: (`Pr.2, Pr.3, Pr.4, Pr.6`). HT4 scores a theory by its *maximum cover*; i.e. the largest percentage of goals found in any world. W_1 contains all the outputs so our economics theory gets full marks: 100%.

HT4 and PIE perform the same task. Both explore the sensitivity of program output to changes in data values at certain locations. Whereas PIE perturbs values at l, HT4 makes assumptions A at l. However, there are differences between the approaches. Firstly, to date, PIE has focused on imperative languages while HT4 has focused on logic programs. However, since imperative programs can be at least partially expressed as theorems or regular expressions (e.g. [18] and Figure 4), this is not a significant difference in the two techniques. Secondly, PIE has no knowledge of output goals G. That is, PIE's D contains only $< Ins, A >$. G can be initialized in many ways. For example, there may exist a database of known or desired behaviour. Alternatively, in the case of specification-based testing, The G needed for HT4 can be initialized from some other theory describing desired behaviours. Also, a human could inspect each input Ins and record what output G is valid (though this will be time-consuming). Further, in the case of $G-$, system constraints can be used to define error cases. Lastly, another option would be to set G^+ to all data points downstream of some mutation l (but heed this warning: this last option would be very slow to execute).

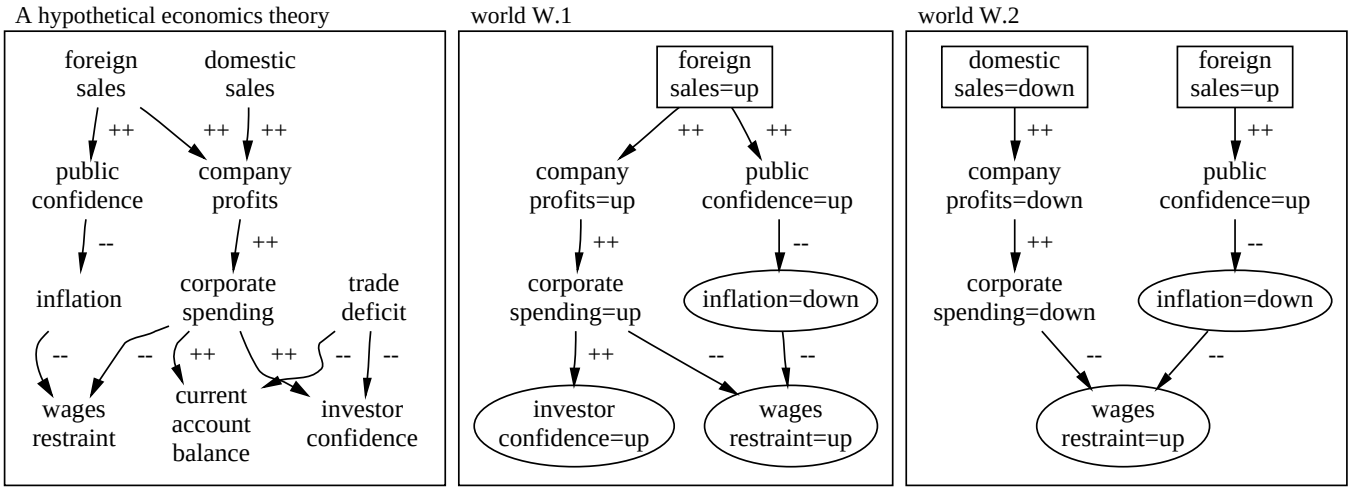

Figure 2. Two worlds from one theory.

HT4 is practical for many real-world theories [7, 15] and has found previously unseen errors in theories in neuroendocrinology published in international refereed journals. Also, it has been shown that one implementation of HT4 was fast enough to manage at least one published sample of real-world expert systems [14]. Nevertheless, the upper bound on HT4-style inferencing is exponential. Selman and Levesque show that even when only one abductive explanation is required and the model is restricted to be acyclic, then abduction is NP-hard [19]. In the specific case of HT4, as it grows proof trees between inputs and goals, no new assignment can be added to a proof that contradicts assignments already in that proof. Finding a directed path across a directed graph that has at most one of a set of forbidden pairs is NP-hard [9] (our forbidden pairs are assignments of different values to the same variable; e.g. the pairs `companyProfits=up, companyProfits=down`).

This worst-case behaviour of HT4 motivated experiments in heuristic optimization. HT4 is an exhaustive search through all explanations. The cost of this exhaustive search is exponential runtimes. What are the comparative benefits of such a search vs a partial exploration of some worlds? To answer this question, HT0 was built. Instead of building all worlds, it builds them one at a time. HT0 tries to force world W_j to be different to $W_1..W_i$ ($i < j$). Further, when little extra is learnt by W_j, HT0 can terminate.

HT0 executes over horn-clauses like Figure 4. The HT0 algorithm is shown in Figure 5. In that figure, square brackets denote ordered sets and the `Permute` function randomly shuffles set order. A persistent store of old runs is maintained in `File`. If `File` already exists, then the best assumptions found to date are retrieved; else they are initialized (line 3). `N1, N2, N3` control the number of searches performed (the variable `Nall` will be used to de-

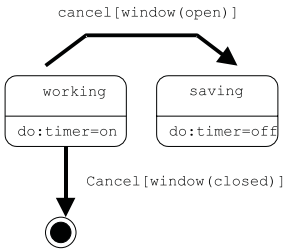

Figure 3. A state transition chart.

```
% Transition #1:
% when working with open documents
% (i.e. window open) and cancelling,
% then goto "saving".
t(saving,t) :-
    t(working,t), t(event,cancel),
    t(window,open).

% Transition #2:
% if working, but no documents open,
% then a cancel takes you straight to "end" .
t(end,t) :-
    t(working,t), t(event,cancel),
    t(window,closed).

% Side effects of entering a state.
t(timer,on)  :- t(working,t).
t(timer,off) :- t(saving,t).
```

Figure 4. Figure 3 as horn clauses.

```
 1 INPUT:  T,Ins, G,  File, N1,N2,N3
 2 OUTPUT: File
 3 (<A,Max>:=readBest(File)) or (A:=[];Max:=0)
 4 Facts := Ins + G;
 5 N1 times repeat
 6 { N2 times repeat
 7   { Covered := [];
 8     T1 := burn(Facts,copy(T))
 9     for Gi in G do
10     { if   <T2,A1>:=thrashBurn(Gi,Ins,T1,A)
11       then {add Gi to Covered
12             A:= A1; T1:= T2;
13     }      }
14     if    size(Covered) > Max
15     then  {Max := size(Covered);
16            append(File,<A,Max,Covered>)
17            if Max=100% then goto :stop}
18     G:=permute(G-Covered)+permute(Covered)
19     A:=change(first(N3,mostUsed(A)));
20   }
21   A := []; G := permute(G);
22 } :stop
```

Figure 5. HT0

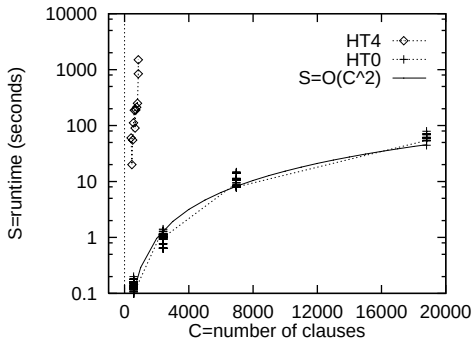

Figure 6. Runtimes

Theory	Clauses	Literals	$\frac{sub-goals}{clause}$	$\frac{clauses}{literals}$
Figure 4	4	8	2	2
T1 [20]	558	273	1.6	2
Random 1	2390	688	1.6	3.5
Random 2	6961	1540	1.7	4.5
Random 3	18803	3394	1.7	5.5

Figure 7. Some theories processed by HT0

note `N1+N2+N3`). `N1-1` times, HT0 clears any old assumptions and randomly permutes the order of the goals (line 21). Then, `N2` times, HT0 tries to prove each goal in order (line 10). `ThrashBurn` is a depth-first search from G_i to any member of Ins across T_1. T_1 is generated (at line 8) from T by `burn`ing away all variable assignments inconsistent with known `Facts` (technically, this is node consistency [11]). As `thrashBurn` searches, if new assumptions are found, they are added to A_1. When a horn clause is accessed, its sub-goals are *thrashed*; i.e. re-arranged randomly. This randomizes the direction of the depth first search from this point on. Also, when a new assumption is made, contradictory assumptions are *burnt* away (i.e. removed via node consistency). The burning and the discovery of new assumptions creates T_2 and A_1 respectively. Lines 11,12 arrange that if G_i is explained, T_2 and A_1 are used for the subsequent searches for G_j ($i < j$). That is, searches for G_j explore a smaller space than G_i. Note that if G_i is explained and G_j is not, then the system does not backtrack to retry G_i. However, we may get another chance to explain G_j since the next time through lines 7-19, we permute the order in which we explore the goals (see line 18). Note, in line 18, when we reset the goal order, we move the uncovered goals to the front of the goal list; i.e. next time through we give priority to things we could not prove this time through. The only other feature of note is line 19. The `N3`-th most used assumptions are changed so that the next time through lines 7-19, the proofs for G are forced into other parts of the theory.

HT0 can be used as an anytime validation algorithm. Assuming that each explanation gives diminishingly less information that the one before, HT0 could run while the programmer traced the `Max` covered value. At anytime, the best `Max` found to date would be available. Also, at anytime, running the system for longer would explore different parts of the theory and (potentially) could find better explanations.

Real world and artificially generated theories were used to test HT0. A real-world theory of neuroendocrinology [20] with 558 clauses containing 91 variables with 3 values each (273 literals) was copied X times. Next, Y% of the variables in one copy were connected at random to variables in other copies. In this way, the theories Random1, Randon2, and Random3 (see Figure 7) were built using `Y=40`. When executed with `Nall` varied from 1 to 50, the $O(N^2)$ curve of Figure 6 was generated. We conclude that HT0 was $O(N^2)$ in these experiments since the R^2 for an $O(N^2)$ curve fit to the HT0 data was 0.98 while the R^2 for $O(N)$, $O(N^3)$, $O(e^N)$ were all < 0.82.

In a result consistent with rapid saturation, no increase in cover was detected above `Nall=5`. That is, (1) the anytime nature of HT0 may not be required since (2) what explanations HT0 can find, it seems to find very quickly.

3 Discussion

HT0 assumes that a small number of random searches performs nearly as well as a thorough search through all combinations of options. This section discusses the generality of that assumption. From many sources, we will conclude that the average size and complexity of the used portions of our programs is much smaller than we might think; i.e. the HT0 assumption is widely applicable.

Menzies and Waugh compared millions of runs of HT4 with HT4-dumb: an HT4 variant that returned any world chosen at random. HT4 out-peformed HT0 by a mere 5.6% (average difference in coverage) [16]. Other experiments show that multiple-world reasoners perform little better than reasoners that reasoners that return one world chosen at random (with some locally guided intelligence regarding what assumptions to make next) [23].

Two studies sampled the operational distribution of one years input to different expert systems. Colomb reports that only a small fraction of the possible state combinations inside his expert system (10^{-11}) was ever executed [5]. Avritzer et.al. found that 99.9% of their 721 input vectors for their system could be represented by a core set of 53 inputs vectors [2].

In a scheduling domain, Crawford and Baker compared inference engines that carefully explored the search space vs theorem provers which lurch across some randomly chosen portion of the space. The random search found more solutions sooner than the careful search [6].

Many running programs only exercise a small portion of their potential total space. Even when we try to explore the entire space of a program, average reachable "objects" (paths, linearly independent paths, edges, statements) is only 40% (and even lower for branch coverage) [8, p302]

Theoretically, there are an exponential number of du-paths: pathways which link where a variable is set to where it is used. However, in practice, a surprisingly small number of program pathways covers all the du-paths [3].

These results, plus the success of HT0, has prompted a mathematical analysis of the odds of reaching a random node across an and-or graph from randomly chosen inputs. Preliminary results suggest that the average reachability odds are very high; e.g. a small number of probes (< 10) has a 99% chance of reaching all that can be reached [12]. Hence, our belief is that saturation-aware inference engines are widely applicable and can optimise testing procedures such as PIE.

Acknowledgements

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

References

[1] A.T. Acree. *On Mutations*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1980.

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

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

[4] T.A. Budd. *Mutation analysis of programs test data*. PhD thesis, Yale University, 1980.

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

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

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

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

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

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

[11] A.K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[12] T. Menzies, B. Cukic, and E. Coiera. Smaller, faster dialogues. In *AAAI'99 workshop on Conflicts and Identifying Opportunities (submitted).*, 1999.

[13] T.J. Menzies. Applications of abduction: Knowledge level modeling. *International Journal of Human Computer Studies*, 45:305–355, 1996. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96abkl1.ps.gz`.

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

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

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

[17] C.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.

[18] K.M. Olender and L.J. Osterweil. Interprocedural static analysis of sequencing constraints. *TOSEM*, 1(2):21–52, 1992.

[19] B. Selman and H.J. Levesque. Abductive and Default Reasoning: a Computational Core. In *AAAI '90*, pages 343–348, 1990.

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

[21] J. Voas and G. McGraw. *Software Fault Injection: Inoculating Programs Against Error*. John Wiley & Sons, 1998.

[22] J.M. Voas. Pie: A dynamic failure-based technique. *IEEE Transactions of Software Engineering*, 18(2):717–727, August 1992.

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

[24] W.E. Wong and A.P. Mathur. Reducing the cost of mutation testing: An empirical study. *The Journal of Systems and Software*, 31(3):185–196, December 1995.