The Complexity of TRMCS-like Spiral Specification

Tim Menzies
Department of Electrical & Computer Engineering, University of British Columbia,
2356 Main Mall, Vancouver, Vancouver, B.C. Canada, V6T1Z4
tim@menzies.com

Abstract

Modern software is often constructed using "spiral specification"; i.e. the specification is a dynamic document that is altered by experience with the current version of the system. Mathematically, many of the sub-tasks within spiral specification belong to the NP-complete class of tasks. In the traditional view of computer science, such tasks are fundamentally intractable and only solvable using incomplete, approximate methods that can be undependable. This traditional view suggests that we should routinely expect spiral specification to always be performed very poorly. This paper is an antidote to such pessimism. Contrary to the traditional view, we can expect that spiral specification can usually be performed adequately, providing that analysts augment their current tools with random probing.

1. Introduction

How hard is our software to specify? Any experienced analyst has horror stories of spaghetti requirements that were fiendishly hard to untangle. Should we routinely expect such horrid cases to be the normal case? In the usual case, just how difficult will it be to to specify our software?

One might expect that as modern software systems grow more and more complex, that it will get harder and harder to specify. For example consider the complexities of specifying a modern distributed information system such as the Teleservices and Remote Medical Care System, or TRMCS[1]. Patients using TRMCS live at home and use automatic monitoring equipment to watch over their condition. In an emergency, patients, or the family or patients, or even the monitors themselves can download patient data to doctors based at medical centers via standard internet lines.

Systems like TRMCS can't be built assuming that every requirement is pre-specified in the requirements phase. Rather, such software must be built in an iterative manner using something like Boehm's spiral model of software development [1]. In the spiral model, version $I+1$ of the system is generated after carefully evaluating a running version I.

This paper uses the term "spiral specification" to refer to the iterative revisions seen in system specifications as it moves through the spiral model. As with the spiral model, the $I+1$ version of the specification will be generated based on issues seen in a running version I. Mathematically, it will be shown below that TRMCS spiral specification contains *NP-complete* sub-tasks. In the view of traditional computer science, such tasks are fundamentally intractable and only solvable using incomplete and approximate methods that may be undependable [24]. Hence, we might routinely expect that spiral specification will be performed very poorly.

Contrary to this standard view, this paper's conclusion is very optimistic about our ability to perform high quality TRMCS spiral specification. It will be shown that within *highly reachability* systems, many of issues within a specification can be extracted using a small number of *non-shallow randomized probes* That is,while in theory TRMCS-like spiral specification is theoretically very hard, it need not be very hard in practice.

The rest of this paper lists the sub-tasks we see in TRMCS-like spiral specification and shows that those tasks are NP-complete. Next, we present experimental and theoretical evidence that we may often assume that our system is highly reachable. Assuming high reachability, we can apply a new style of cost-effective minimal analysis. In this new minimal style, when faced with uncertainty or specification issues, analysts can quickly explore their systems for a limited time using a randomized search engine. Assuming high reachability, then such a randomized search should quickly discover all the important features of that specification.

[1]Described at `http://www.ics.uci.edu/iwssd/case-study.pdf`.

2. Some Sub-Tasks of Spiral Specification

We will infer the complexity of spiral specification from the analysis of its sub-tasks. This section lists some of those sub-tasks. This list is hardly complete but TRMCS-style spiral specification is at least as hard as the sum of the complexity of the following sub-tasks.

In spiral specification, the next version of the system is created via experience with the current version. The more we can exercise the current version, the more issues we can see and the more we can improve it. Hence, the complexity of the runtime tasks of TRMCS contributes to the complexity of watching and improving the system. Three such runtime sub-tasks within TRMCS are *diagnosis*, *planning*, and *monitoring*. Diagnosis tools are required to define treatment regimes for sick patients. TRMCS will also need to generate multiple alternate plans. Such multiple plans are required to find the most cost-effective method for delivering health care. Once a plan is selected, we need to monitor that plan. When new data arrives from the patient's monitoring equipment, we need to always check that our current plan is not invalidated by some new circumstance. For example, an elevated temperature might imply an new infection that requires different antibiotics.

TRMCS specification could well require *reasoning in the presence of inconsistencies*. With the spiral approach, specifications may not be written by a single author at a single time. Hence, the odds of new specs contradictory existing specs is non-zero. In fact, we can almost guarantee that TRMCS specifications will contain contradictions due to the competing aspects of the system:

- Family members may seek to *maximize* the amount of monitoring equipment while patients may seek to *minimize* the expense of their illness to their family.
- The TRMCS medical centers who, according the current specification, must compete to service medical emergencies. Competitions must be judged and the judging critieria will probably be keenly debated,
- The treatments proposed by the medical centers must *maximize* for patient health while *minimizing* the required health resources such as unnecessary calls to ambulance services, the prescription of overly expensive drugs, or the the use of inappropriate surgical procedures.

Proponents of formal logics such as van Lamsweerde [25] argue that inconsistencies should be removed once detected. Proponents of non-standard logics such as Nuseibeh argue convincingly that we should retain our consistencies since they record the differences between our stakeholders [18]. The premature resolution and removal of inconsistencies can alienate a stakeholder group [7] Further, without access to the requirements of a particular stakeholder group, inappropriate future design decisions could be made.

Safety critical systems such as TRMCS require *validation*. TRMCS must meet certain certifications standards such as 24x7 availability, real time response rates, monitoring accuracy of the equipment, ensuring that at least one medical center will service the emergency, and assessing the quality of care offered by each center. Further, in the TRMCS case, such certifications must be made for a system that may:

- Contain inconsistencies;
- Be only a partial prototype of the full system;
- Include cost cutting heuristics and artificially intelligent diagnosis algorithms;
- Change when patients require different levels of monitoring (e.g. they are diagnosed with new diseases) or buy new equipment, when a new treatment becomes available, or when the specification changes.

3. Analyzing Sub-Task Complexity

The complexity of TRMCS-style spiral specification is at least as complex as the complexity of the sub-tasks of diagnosis, planning, reasoning in the presence of inconsistencies, and validation. This section explores the complexity of these sub-tasks and, hence, of spiral specification.

To compute the complexity of diagnosis, planning, etc, we map these tasks into *logical abduction*. Abduction belongs to the *NP-complete* class of tasks and, hence, so to does spiral specification.

This mapping from spiral specification to the NP-complete tasks leads a very negative result concerning the complexity of spiral specification; i.e. it is as fundamentally intractable as every other NP-complete task. The rest of this section defines abduction and NP-completeness, and maps our sub-tasks to abduction.

Note that our sequel will reverse this pessimism of this proof of NP-completeness.

3.1. Introducing Abduction

Logical abduction can support the sub-tasks of TRMCS-style spiral specification described above. Abduction is often informally defined as inference to the best explanation [19]. This definition often leads to the mistaken belief that abduction is only useful for explanation. This is not correct. Dozens of applications of abduction are listed in [10, 12]. For a specific discussion on the applications of TRMCS, see below.

Abductive inference builds a world of belief W_i from a theory T:

$$W_i \subseteq T \quad (1)$$

The world W_i makes assumptions A_i that allow us to achieve some goals G:

$$W_i \cup A_i \vdash G \tag{2}$$

While the theory T_i may contain inconsistencies, the generated world must not:

$$W_i \cup A_i \not\vdash \perp \tag{3}$$

That is, a world W_i is the useful part of a theory T which, when combined with assumptions A_i, lead to some goal G (Formula 2) without causing any contradictions (Formula 3).

If multiple such worlds exist, then a domain specific BEST operator is used to return the preferred world(s).

$$\begin{aligned} W_{best} &= BEST(W) && (4) \\ W_{best} &\subseteq W && (5) \end{aligned}$$

Abduction can reason in the presence of inconsistencies. It does this by finding the consistent islands within a theory containing inconsistencies. This is a useful tool for (e.g.) exploring conflicts between competing stakeholders. Suppose we know who wrote each portion of a specification. Significant conflicts between users can be detected when different people's ideas end up in different worlds of belief. These conflicts can then be negotiated by focusing on the key conflicting assumptions that split the worlds. Disputes between feuding users can be adjudicated as follows:

- Focus only on the key disputes that divide the worlds;
- Declare that the winning user offers theories which build worlds that can explain most of the desired behavior [16, 26].

Abduction also support validation, diagnosis, planning, and monitoring:

- Abductive validation asks the question "What is the maximum percentage of known/desired behavior that can be explained by some theory?" This can be implemented using a BEST that favors worlds with the largest intersection to some output goal set. Abductive validation has found previously undetected faults in theories containing inconsistencies; e.g. qualitative theories of neuroendocrinology [11, 13, 14].
- Many different approaches to diagnosis have previously been mapped to a single abductive procedure (see the discussion in [5]). For example, minimal fault diagnosis can be implemented using a BEST that favors worlds with the fewest possible inputs and the most known goals [20].
- Abductive planning uses a BEST that favors worlds with the most goals and the least total cost (e.g. least number of surgical interventions).
- Abductive monitoring means caching the worlds generated by abductive planning, then deleting any world whose assumptions contradict any newly arriving data. Whatever worlds remain in the cache represent the space of possible futures known to the system.

3.2. NP-Complete Tasks

Abduction can be mapped to a class of tasks known as the NP-complete tasks. No fast and complete algorithm has been found for the NP-complete tasks, despite decades of research. Therefore, by showing a connection from spiral specification to NP-complete tasks, we also have a strong theoretical reason for doubting our ability to reasoning thoroughly during the spiral specification process. The rest of this section shows that abduction is NP-complete

Problems are proved to be NP-complete if we can *reduce* (i.e. transform) that problem quickly (i.e. polynomial time) to a known NP-complete problem. Such NP-complete problem can be solved by non-deterministic algorithm which may find an answer slowly (in exponential time, worst case) but the solution can be verified quickly (in polynomial time). The upper bound on the runtime of NP-complete tasks is exponential on problem size. Moore's Law tells that computers are speeding up at an exponential rate. However, this speed up is far too slow to catch up with the worst case exponential runtimes of NP-complete tasks. Hence, in the view of standard computer science, if a problem is NP-complete then the best we can hope for is approximate solutions of indeterminable quality.

We can reduce abduction to an NP-complete tasks using a simple result from 1976 (for other reductions, see [3, 21]). Gabow et.al. [8] showed that building pathways across programs with impossible pairs (e.g. some boolean and its negation) is NP-complete for all but the simplest programs[2]. Abduction can be mapped to building pathways across programs with impossible pairs. Consider the case of testing Formula 2 and Formula 3 using a proof procedure that builds proof trees across logical rules. Rules can share terms; e.g. a post-condition in one rule may be found in the pre-condition of another rule. This sharing of terms can be viewed as a directed graph; e.g. Figure 1. Note the "no-edges" in Figure 1: our rules contain terms that may contradict other terms. These no-edges are pairs of incompatible nodes; i.e. the Gabow pre-condition for NP-completeness. Any procedure that builds a proof tree across Figure 1 (e.g. to demonstrate Formula 2) which must continually test for these forbidden pairs (e.g. to demonstrate Formula 3) is hence NP-complete.

[2]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.

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

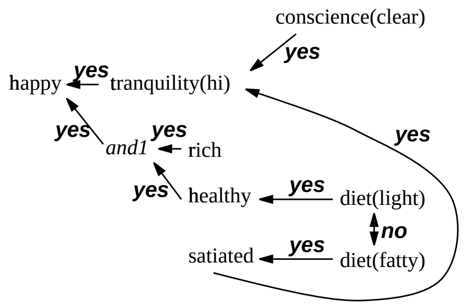

Figure 1. A graph connecting terms within some horn clauses.

3.3. Summary

Since spiral specification sub-tasks have an abductive mapping, and abduction is an NP-complete task, then spiral specification is NP-complete. That is, in the traditional view of complexity theory, spiral specification is fundamentally intractable and only possible using incomplete, approximate methods that can be undependable. Hence, in the traditional view, we would routinely expect spiral specification to always be performed very poorly.

4. Searching an Indeterminate Space

This section is an antidote to the pessimism offered above. Empirical and theoretical results strongly suggest that, in the usual case, randomized searching of this space of incompatibilities is not as hard as suggested by Gabow et.al. This result will suggest a new minimal style of specification analysis. In this new style, when faced with uncertainty or specification issues, analysts can quickly explore their systems for a limited time using a randomized search engine.

4.1. Empirical Results

A curious experimental result is that theoretically slow NP-complete tasks can be completed adequately in a very short time. This section describes some of those experiments. The next section argues that these experiments demonstrate a general principle; i.e. NP-complete tasks such as spriral specification can be performed adequately and quickly using using randomized search.

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

Figure 2. Average performance of elaborate search (TABLEAU) vs randomized search (ISAMP) on 6 scheduling problems (A..F) with different levels of constraints and bottlenecks. From [6].

```
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;
          give v a randomly chosen value;
          forward_chain_from(v);
          if contradiction exit loop;
    fi
   }
  } return failure
```

Figure 3. The ISAMP algorithm. From [6].

Figure 2 shows Crawford and Baker's comparison of a standard depth first search backtracking algorithm (TABLEAU) to a randomized search theorem prover (ISAMP) [6]. Figure 3 describes how 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`.

Experiments with randomized abductive world generation showed that exploring a few worlds returned nearly as much information as a rigorous exploration of all worlds. The HT0 abductive algorithm [17] in Figure 4 just returns the first world it randomly finds using the process described in Figure 5. In contrast to HT0, HT4 [11] carefully constructs every possible world. HT4's runtimes were observed

```
% Test ors/amds in random order
X ror  Y :- maybe -> (X;Y); (Y;X). %or
X rand Y :- maybe -> (X,Y); (Y,X). %and
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(10ˆ3)/10ˆ6,
    (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]).
```

Figure 4. 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 [17]. For brief notes, see Figure 5.

HT0 tries to prove a list of sorted goals. The assumptions made when trying to proving goal i must not contradict the assumptions made when proving goals $1 \ldots (i-1)$. 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. 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 $i+1$. 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.

Figure 5. Explanation of the HT0 code in Figure 4.

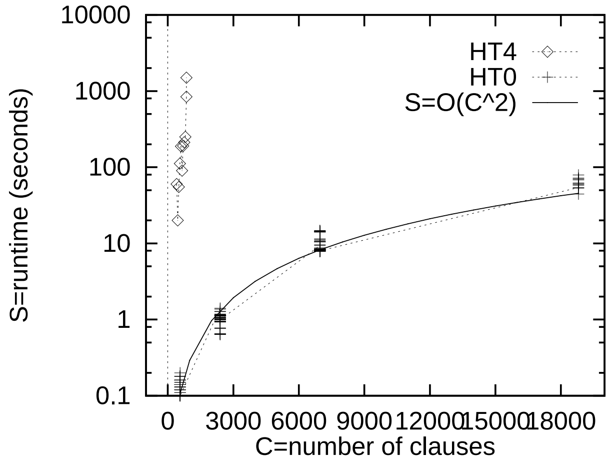

Figure 6. Runtimes: HT4 vs HT0

to be exponential: $O(2^N)$. We should expect such exponential runtimes from algorithms performing NP-complete tasks such as abduction. Surprisingly, the random search of HT0 performs nearly as well as the more thorough HT4 algorithm. Further, when running on the same problems as HT4, HT0’s runtimes were observed to be much faster ($O(N^2)$: see Figure 6). In the region where the two algorithms terminated on the same problems, HT0 found worlds that covered 98% of the outputs found in the HT4 belief sets. In other experiments with an earlier version of HT0, Menzies, Easterbrook, Nuseibeh and Waugh compared how arguments resolve within a specification [16]. HT0 style reasoning was used to find one argument resolution, picked at random. These results were compared to resolutions generated from the all-worlds search of HT4. The average difference between the two searches was less than 6%.

The adequacy of randomized search has also been seen in extensive empirical studies by the constraint satisfaction community; e.g. [4, 9, 23]. Searching a space of conflicting constrains is NP-complete and therefore theoretically very slow. However, experience shows that such NP-complete tasks are only truly slow in very narrow regions. For example, Figure 7 shows the number of times a particular NP-complete algorithm hits a dead end and has to backtrack. Outside of a narrow zone, the algorithm could terminate quickly without extensive backtracking. The slow zone corresponds to the *phase transition* zone between an under constrained zone and a over constrained zone. In the over constrained zones, the odds of finding a solution are very low. Further, if the over constraints are very tight, we can quickly discover that no solution exists since our searches are all quickly blocked. Figure 7 is over constrained above $X = 6$. In an under constrained problem, the odds of find a solution is very high since many solutions exist. Fig-

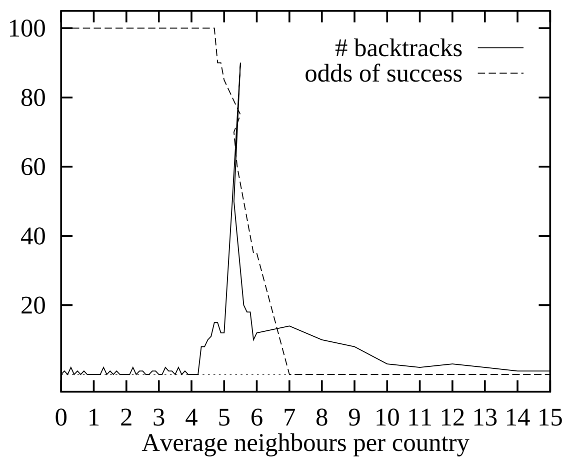

Figure 7. A phase transition effect for map coloring (no two adjacent countries on a map should be the same color); adapted from [4].

ure 7 is under constrained below $X = 5$. Note that under constrained problems can be solved by simple methods and over constrained problems can't be solved by any method[3] Hence, for problems outside the phase transition zone, simple search engines can solve problems. For example, a random initial selection of variables, following by some cheap inferencing , is often used; e.g. ISAMP, HT0, and GSAT [22]. GSAT has proved theorems in knowledge bases one to two orders of magnitude bigger than ever done before.

4.2. Theoretical Results

The above case studies suggest the average cost of Formula 3 is far less than the worst case cost. If this was a general result, then software project managers would have greater confidence in the practicality of TRMCS-like spiral specification. This section explores this issue of generality and makes the following counter-intuitive conclusion:

> On average, a few randomly selected abductions will find most of the reachable features of a theory, even in theories containing inconsistencies.

With Cukic and Singh, I have built an average case *reachability* model that finds the average probability of reaching a random node in a dependency graph, given "in" number of randomly selected inputs [15]. This reachability model is a virtual machine for abduction. That is, from a space of possibly contradictory ideas, reachability theory finds the average odds of reaching some goal without causing contradictions; i.e. Formula 2 and Formula 3.

[3]Exception: if certain constraints are not hard constraints, (i.e. they can be relaxed) then some kind of over constrained problems can be solved [2]. In terms of Figure 7, such relax-able constraints would drive a problem into the solvable zone.

Dependency graphs can be generated from many systems; e.g.

- Figure 1 showed the translation of some horn clauses into a dependency graph.
- Finite state diagrams can be reduced to horn clauses [15] and hence can be expressed as dependency graphs. Finite state diagrams can be found in many analysis methods or can be automatically derived from a static analysis of a program

We will say that dependency graphs have V nodes which are divided into and-nodes and or-nodes with ratios $andf$ and orf respectively ($orf + andf = 1$). Recalling Figure 1, we note that these nodes are connected by yes-edges (which denote valid inferences) and no-edges (which denote the inconsistencies). On average, each node is touched by no_μ no-edges. When inferencing across these graphs, variables can be assigned, at most T different values; i.e. a different assignment for each time tick T in the execution (classic propositional systems assume $T = 1$; simulation systems assume $T \geq 1$). Nodes are reached across the dependency graph using a network of height j where the inputs are at height $j = 0$. This network represents W_i of Formula 1 and Formula 2.

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 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:

$$i = \beta(depth) * (j - 1) \tag{6}$$

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.

$$P[j]_{and} = P\left[j - 1\right] * \left(\prod_{2}^{andp[j]} P[i]\right) \tag{7}$$

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.

$$P[j]_{or} = 1 - (1 - P\left[j - 1\right]) * \left(\prod_{2}^{orp[j]} (1 - P[i])\right) \tag{8}$$

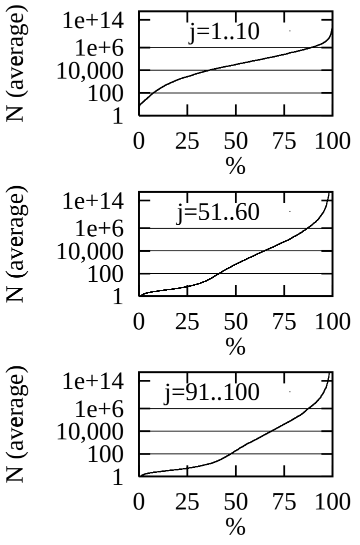

Figure 8. Some frequency distributions of the number of tests required to be 99% sure of reaching a node at height j generated from the Menzies-Cukic-Singh reachability model.

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.

$$P[j] = andf[j] * P[j]_{and} + orf[j] * P[j]_{or} \qquad (9)$$

Other details not shown above are loop detection and the modeling of Formula 3 (contradiction detection). See [15] for the full details.

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 randomly selected inputs. N randomly selected inputs has certainty

$$C = 0.99 = 1 - \left((1 - P[j])^N\right) \qquad (10)$$

of reaching a node in dependency graph. Hence,

$$N = \frac{log(1 - 0.99)}{log(1 - P[j])} \qquad (11)$$

The above model was run for a wide range of the above parameters; e.g. up to 10^8 nodes, up to 1000 inputs, up to 100 time ticks, wildly varying the frequency and skew of and-nodes, or-nodes, and no-edges, etc. The frequency distribution of the calculated N values is shown in Figure 8 divided according to the j (proof height) value.

The simulation results shows that much of the reachable parts of a system can be reached very quickly using randomized probing. After searching into a theory for more than a shallow depth (e.g. $j \geq 50$), nearly half the nodes can be reached with probability of 99% using less than 100 randomly selected inputs. Further, tens to hundreds of thousands of randomly selected inputs will reach nearly all the nodes. Note that these runs can be performed in parallel. Given that the standard desktop machine is now a 500MHz box, then in many situations, organizations can perform enough random inputs to implement Formula 3.

What is surprising about this result is that it holds for supposedly exponentially slow NP-complete tasks like spiral specification. Recall that in this model, we search across a dependency graph contains no-edges. These no-edges connect incompatiable pairs of nodes. The Gabow et.al. result suggests that buidling a proof tree across this dependency graph should be very slow (since that task is NP-complete). However, on the contrary, the simulation results from the model show that randomized search can very quickly reach all that can be reached even for this NP-complete task.

5. Discussion

If we manually true to understand all parts of a complex specifications, much of their intricacy will escape us. In the case of the TRMCS system, that understanding implies performing the sub-tasks of reasoning in the presence of inconsistencies, validation, diagnosis, planning, monitoring, and validation. This sub-tasks belong to a class of tasks which, in the worst case, is intractable.

However, if we automatically explore complex specifications using randomized probes, then on average we will find out most of what can be found within those specifications. This is a counter intuitive result (to say the least!). Our pre-experimental intuition was that searching a complex space containing inconsistencies would be very hard indeed. However, if thoroughly exploring all the consequences of inconsistencies in a theory were as hard as predicted by the theory of NP-completeness, then we should expect systems like ISAMP, GSAT, and HT0 to perform very poorly. They don't. Hence, a small number of random searches around inconsistent theories will sample that system as well as some other non-random strategy.

This suggests a new style of specification analysis. After some initial manual tinkering, analysts should connect some machine readable form the specifications to a Monte Carlo simulator (which can generate random inputs) or a random search engine like HT0 (which can perturb the internal execution of the system in many different directions). Such randomized search procedures should reveal most of what can be revealed, even in complex specifications.

Acknowledgements

Helen Burgess provided invaluable editorial advice for this article. This work was partially supported by NASA through cooperative agreement #NCC 2-979.

References

[1] B. Boehm. A spiral model of software development and enhancement. *Software Engineering Notes*, 11(4):22, 1986.

[2] A. Borning, M. Maher, A. Martindale, and W. Wilson. Constraint hierachies and logic programming. In *Proceedings of Sixth International Logic Programming Conference Lisbon, Portugal*, pages 149–164, 1989.

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

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

[5] L. Console and P. Torasso. A Spectrum of Definitions of Model-Based Diagnosis. *Computational Intelligence*, 7:133–141, 3 1991.

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

[7] S. Easterbrook. Handling conflicts between domain descriptions with computer-supported negotiation. *Knowledge Acquisition*, 3:255–289, 1991.

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

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

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

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

[12] T. 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`.

[13] T. 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`.

[14] T. 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`.

[15] T. Menzies, B. Cukic, H. Singh, and J. Powell. Testing indeterminate systems, 2000. ISSRE 2000.

[16] 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://research.ivv.nasa.gov/docs/techreports/1999/NASA-IVV-99-009.pdf`.

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

[18] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging inconsistency in software developoment. *IEEE Computer*, 33(4):24–29, April 2000. Available from .

[19] P. O'Rourke. Working notes of the 1990 spring symposium on automated abduction. Technical Report 90-32, University of California, Irvine, CA., 1990. September 27, 1990.

[20] J. Reggia, D. Nau, and P. Wang. Diagnostic Expert Systems Based on a Set Covering Model. *Int. J. of Man-Machine Studies*, 19(5):437–460, 1983.

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

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

[23] B. Smith and M. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.

[24] R. R. T.E. Cormen, C. E. Leiserson. *Introduction to Algorithms*. MIT Press, 1990. ISBN: 0262031418.

[25] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings ICSE2000, Limmerick, Ireland*, pages 5–19, 2000.

[26] D. Zowghi, A. Ghose, and P. Peppas. A framework for reasoning about requirements evolution. In *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence (PRICAI96), Cairns, Australia, August*, 1996.