Machine Learning for Requirements Engineering

Tim Menzies[1,2,3]
University of British Columbia
tim@menzies.com

James D. Kiper[4]
Miami University
kiperjd@muohio.edu

Ying Hu[3]
University of British Columbia
huying_ca@yahoo.com

ABSTRACT

In practice, requirements engineering is complicated by the valid interests of multiple stakeholders. Meeting these varied and sometimes conflicting needs of multiple stakeholders is the primary task of requirements engineering. One method of helping requirements engineers in this task is to treat requirements as a theory, then to write simulators to exercise that theory. This exploration of the requirements theory can help identify the leverage points and critical differences among stakeholder views. This can then be used to generate insightful debates among these human stakeholders.

Such requirements theory simulators can generate an overwhelming amount of data. Machine learners can summarize this excess of data and report the key features of a simulation. In our case study, the TAR2 machine learner uses a model of CMM level-2 to find a minimum set of changes to a software project that increase the likelihood of low cost, better projects.

KEYWORDS: knowledge acquisition tools, knowledge engineering and modelling methodologies, abduction, machine learning, requirements engineering, stochastic search, treatment learning, CMM level-2.

1. INTRODUCTION

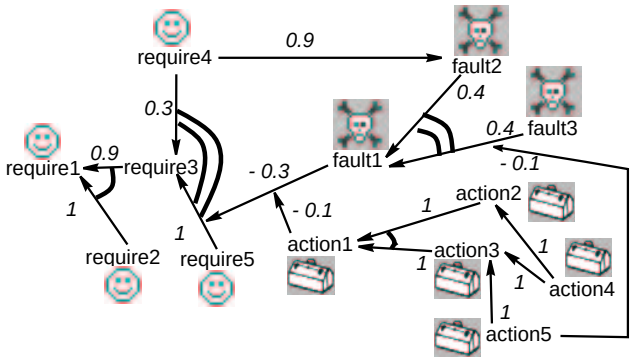
Requirements engineering (RE) is (1) the elicitation of high-level goals of some envisioned system followed by (2) the refinement of these goals into services and constraints, and (3) the assignment of responsibilities for the resulting requirements to agents such as humans, devices, and software [26]. RE is typically performed within a community of stakeholders, who may have different goals and priorities.

Tools that support RE should (a) assist the stakeholders in finding the key disputes, (b) find trade-offs between the stakeholders that address those disputes; (c) discourage debates over issues with little impact to the overall goals, and (d) refine and evaluate the early life cycle models. The intent of RE tools is to generate insightful debates between human stakeholders. Hence, it is both misguided and impossible for RE tools to fully automate the RE process since such total automation would prevent human participation and human acceptance of the conclusions of the debate.

Model-based RE converts English requirements documents into some formal theory. For example, Mylopoulos et.al. use *goal-graphs* which are directed and-or graphs of literals [19]. Van Lamsweerde's work focuses on the *product aspects* of a system. The core data structure in that work is a *goal hierarchy* where every node contains assertions about products in the domain, expressed as fragments in the UML syntax. As system goals change, parts of this hierarchy are pruned and the implemented system is designed from the remaining UML fragments [26]. The ARRT tool focuses on the *process aspects* of a system [5].The core data structure in ARRT is a network of *faults* and risk mitigation *actions* that effect a tree of *requirements* written by the stakeholders (e.g. see Figure 1). Potential faults within a project are modelled as influences on the edges between requirements. Potential fixes are modelled as influences on the edges between faults and requirements edges. Each edge in the network is augmented with a numeric strength. These numerics quantify the relationships among requirements, risks, and the activities we use to mitigate these risks. ARRT-style RE outputs a minimum set of risk-mitigation "control actions" which address the potential faults.

The research described here began with the speculation that the Bratko et.al. process of "simulation+summarization" could augment ARRT-style RE. Bratko et.al. [1] use machine learning to understand theories. Given a domain theory expressed as horn clauses, they use a resolution-based simulator (some Prolog program) to generate a log of possible behaviors from the theory. This log is then given to some machine learner to generate a summary of the essential features of a theory. This summary must be useful for RE. For example, the the RE control problem studied here is that of *treatment learning*; i.e. learn *minimum treatments* from the behavior logs that improve the overall behavior of a system. Suppose a log entry lists the values seen in the theory at-

[1]Submitted to the 16th IEEE International Conference on Automated Software Engineering Nov 20-22, San Diego, USA. http://sigart.acm.org/Conferences/ase/. WP ref: 01/ml4re/ June 29, 2001
[2]Corresponding Author. Phone: (604) 822-3381 Web: http://tim.menzies.com.
[3]Address: Department of Electrical & Computer Engineering; 2356 Main Mall; Vancouver, B.C. Canada V6T1Z4.
[4]Dept. Computer Science & Systems Analysis, Miami University, Oxford, Ohio, USA.



Faces denote requirements;

Toolboxes denote actions;

Skulls denote faults;

Conjunctions are marked with one arc; e.g. *require1* if *require2* and *require2*.

Disjunctions are marked with two arcs; e.g. *fault1* if *fault2* or *fault3*.

Numbers denote impacts; e.g. *action5* greatly reduces the impact of *fault3*, *fault1* reduces the impact of *require5*, and *action1* reduces the negative impact of *fault1*.

Figure 1: Left: an An requirements model. Right: explanation of symbols.

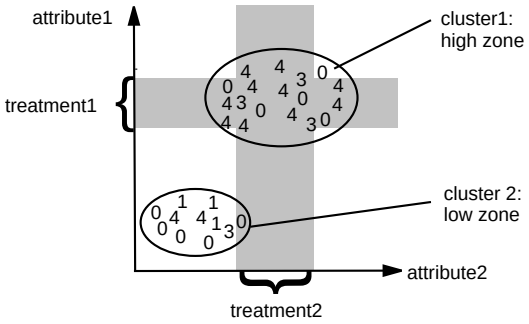


Figure 2: Treatments in attribute space.

tributes during a run; e.g. $attributeX = 3.1, attributeY = 3.2$. The worth of such a log entry is scored according to some domain-specific predicate; e.g.

$$worth \leftarrow score(AttributeX, AttributeY)$$

For example, Figure 2 shows the assigned scores generated from 29 entries in a log. If the control attributes could be constrained into the two treatments shown in Figure 2, then subsequent executions of the RE theory would yield a higher ratio of the more desired behaviors (those that get scored with a higher worth). Note that a prerequisite for treatment learning is that some attributes are *control variables*; i.e. they have some causal influence on the system.

The rest of this paper explores learning treatments via the simulation and summarization of RE theories. Bratko et.al. argue that off-the-shelf machine learners are suitable for the summarization. Our experience does not support that claim, at least for the theories we studied and the software managers we work with. Firstly, the problem shown in Figure 2 is subtlety different from the standard machine learning problem solved by (e.g.) C4.5 [22]. Standard learners output some descriptor of the clusters in attribute space. Our ideal RE learner learns *difference between* clusters. Secondly, as discussed in §2.1, several issues complicate the simulation of RE theories. Thirdly, as argued in §2.2, standard learners such as C4.5 can be unsuitable for summarizing the logs from the simulation of nondeterministic RE theories. We argue in §3 that if the topology of RE theories contain *narrow funnels* (defined below) then simpler simulation and summarization can be implemented. Assuming narrow funnels, then simple learners like the TAR2 system discussed in §4 can solve the treatment learning problem for RE theories. §2.1 to §3 discuss past work. §4 is the novel contribution of this paper.

2. OPEN ISSUES

2.1 Simulation Issues

Five problems complicate the process of simulating RE theories. Firstly, the theories themselves may be open to debate. For example, in the above sample of a ARRT-style RE model, the magnitude of the influence strengths may be unknown with certainty.

Secondly, Bratko et.al. exhaustively executed their models and this proved impractical in our test domains. For example, Menzies & Sinsel reported a domain where exhaustive execution of an RE theory, plus all combinations of the control actions, needed 10^9 runs [14].

Thirdly, in the Menzies & Sinsel domain, experiments with random sampling within the 10^9 possibilities yields decision trees with tens of thousands of nodes. Such tree are too large for human comprehension and must be further summarized.

Fourthly, the simulator itself can be non-trivial to implement. RE theories are generated very early in the software life cycle and may contain contradictions (especially if they are built by a community of competing stakeholders). Simulators for RE theories must be contradiction tolerant. A general framework for contradiction-tolerant inference procedure is *abduction*, as used in HT4 [17] and truth maintenance systems such as the ATMS [4]. (the connection of TMS to abduction is discussed in [2]). Abduction seeks assumptions that lead to goals from inputs without causing contradictions. A consistent set of assumptions defines a *world* of belief. Intuitively, each world is a set of ideas that can be believed at the same time. Even simple abductive inference engines can be complex to build [16].

A fifth problem with simulating an RE theory is that such a simulation can take exponential time and memory. Gabow et.al. showed that building proofs across theories containing contradictions (e.g. $\{x, \neg x\}$) is NP-hard for all but the simplest theory (a theory is very simple if it is very small, or it is a simple tree, or it has a dependency networks with

abbreviations		current situation	proposed changes
prec = 0..5	precedentness	0, 1	
flex = 0..5	development flexibility	1, 2, 3, 4	1
resl = 0..5	risk resolution	0, 1, 2	2
team = 0..5	team cohesion	1, 2	2
pmat = 0..5	process maturity	0, 1, 2, 3	3
rely = 0..4	required reliability	4	
data = 1..4	database size	2	
cplx = 0..5	product complexity	4, 5	
ruse = 1..5	level of reuse	1, 2, 3	3
docu = 0..4	doco requirements	1, 2, 3	3
time = 2..5	runtime constraints	?	
stor = 2..5	main memory storage	2, 3, 4	2
pvol = 1..4	platform volatility	1	
acap = 0..4	analyst capability	1, 2	2
pcap = 0..4	programmer capability	2	
pcon = 0..4	programmer continuity	1, 2	2
aexp = 0..4	analyst experience	1, 2	
pexp = 0..4	platform experience	2	
ltex = 0..4	experience with tools	1, 2, 3	3
tool = 0..4	use of software tools	1, 2	
site = 0..5	multi-site development	2	
sced = 0..4	time before delivery	0, 1, 2	2
	# of combinations=	$6 * 10^6$	

Figure 3: A NASA software project. Unknowns in the current situation are shown as ranges or, in the case of total lack of knowledge, a "?".

out-degree ≤ 1) [6]. Subsequent research reached the same pessimistic conclusion (see the review in [17]). For example, both the ATMS and HT4 exhibited runtimes exponential on theory size [17]. In order to tame the runtime problem, RE analysts have imposed severe restrictions on the nature of their theories or the questions being asked of that theory (e.g. [13]). Our goal is to permit such extensions, without requiring slow or complex processing.

2.2 Summarization Issues

Initially it was hoped that off-the-shelf machine learners could summarize logs generated from RE theories. Initial experiments with treatment learning using TAR1 system were encouraging [14]. However, the drawbacks with TAR1 discussed below prompted the development of TAR2.

Menzies & Sinsel studied a log of 50,000 examples generated from a COCOMO-based risk evaluation tool [8] applied to the NASA project shown in Figure 3. That tool required a guesstimate of the source lines of code (SLOC) in the system and certain internal tuning parameters which, ideally, are learnt from historical data. Lacking such data, Menzies & Sinsel used three guesses for SLOC and three sets of tunings which they took from the literature. Competing stakeholders proposed 11 changes to a project. Some of the project features were unclear and, for those features, project managers could only offer ranges for the required inputs to the COCOMO-based tool. These ranges offered 2930 possible combinations for the inputs. When combined with the other uncertainties, this generated a space of 54

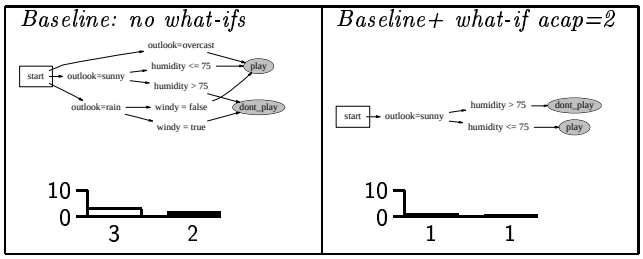


Figure 4: TOP: A decision tree (left) and a pruned tree (right) holding all branches that do not contradict *acap=2*.
BOTTOM: Number of branches to different risk classification. Legend: ☐ =low risk ■ =high risk.

million possibilities ($2930 * 2^{11}$*three guesses for SLOC * three tunings).

Faced with this overdose of possibilities, Menzies & Sinsel performed 50,000 Monte Carlo simulations where the inputs were taken from the 54 million possibilities. A machine learning program generated decision trees from the 50,000 runs. A tree query language called TARZAN then swung through the learnt trees looking for the least number of attribute ranges that had the biggest impact on the overall software development risk. TARZAN treated the learnt trees as a space of possibilities within the logged behaviour. TARZAN what-if queries by pruning all branches in the learnt trees that contradict our what-if possibility[1]. For example, if we wonder "what-if acap=2", then Figure 4, top left, would be pruned to Figure 4, top right. This particular "what-if" turns out to be a bad idea. The histograms in Figure 4, bottom, show that this pruning drives us into a situation where the ratio to low risk to high risk projects changes from 3:2 to 1:1. That is, if *acap=2*, then we increase our chances of a high-risk project.

Figure 5 shows some of the what-if queries conducted over the trees learnt from the 50,000 runs. The baseline risk profile is shown in cell A1 of Figure 5: prior to the what-if queries, the learnt trees hold branches to 7,24,8 low,medium,high risk projects respectively. Seven of the proposed changes had little impact on the baseline. Of the remaining four proposed changes, two are clearly superior. Cell A2 shows that that having moderately talented analysts and no schedule pressure (*acap=[2], sced=[2]*) reduced the risk in this project nearly as much as any other, larger subset. Exception: B2 applies actions to remove all branches to medium and high risk projects. Nevertheless, Menzies & Sinsel recommended A2, not B2, since A2 seemed to achieve most of what B2 can do, with much less effort.

Note that Figure 5 takes $\frac{1}{6}$th of a page to display and shows the key factors that control the classifications of 54,000,000 possibilities. This astonishing reduction in the argument space is consistent with the COCOMO-based tool containing narrow funnels.

[1]The fanciful name of TARZAN arose when it was realized that these what-if queries are like benevolent agents swinging through the trees looking for ways to change what is going on. Tools that extend TARZAN should come from the same genre. Hence, JANE and CHEETAH.

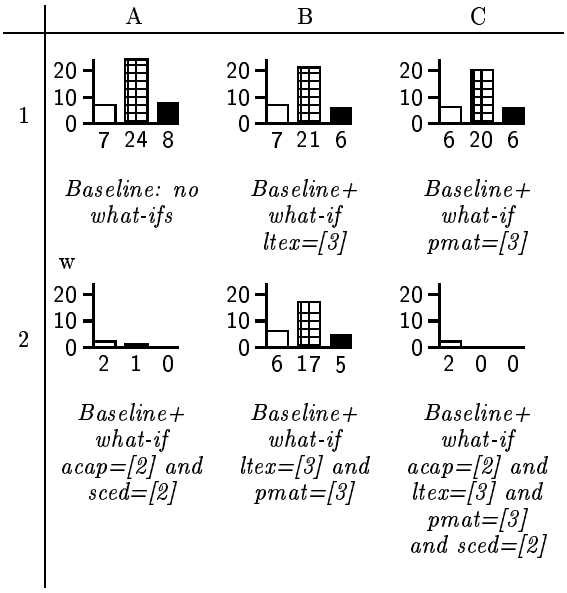


Figure 5: Number of branches to different risk classifications. Legend: =low risk =medium risk =high risk.

While this experiment was encouraging, the technique had a major drawback for summarizing general RE theories. The COCOMO system is a well developed deterministic theory. Each run assigns values to the same attributes. A nondeterministic abductive simulation may not reach the same variables during each run and the datasets collected from such a run may have a very large number of unknown values. For example, in one example we studied, 45% of the attributes were unknown in each vector. This leads to very large error rates in theories learnt from C4.5 and very unreliable treatments. Hence, we concluded that decision tree learners like C4.5 were unsuitable for summarizing the logs of a nondeterministic simulation.

3. FUNNEL THEORY

To understand our resolution to the above issues, we first must introduce *funnel theory*. Funnel theory is a claim that, on average, the number of goals reached via the generation of proof trees from inputs to goals is not greatly effected by nondeterministic search. As we shall see, this claim simplifies both the simulation and summarization of RE theories.

Consider a proof tree that connects inputs to goals. In the HT4 framework, a node in a proof tree is either a facts or an assumption. (In HT4, a node is an assumption if it is not a member of the inputs or goals.) We assume that a single tree is consistent (i.e. it does not contain contradictory assumptions) but trees may contradict other trees (if one tree assumes something that contradicts the assumptions of another tree). Depending on which consistent subset of the assumptions is believed, different sets of proof trees will be believed and different goals will be reachable.

After discarding the non-contradictory assumptions, and the assumptions that can be proved from other assumptions, a set of minimal contradictory assumptions can be found with a proof tree that control the assumptions made in the rest of the tree. This union of this set across all known proof trees is the funnel. If the average size of this set is very small then the total number of distinct sets of consistent proof trees that can be generated is very small.

Research analogous to funnel theory has been reported in many domains (see §6). The funnel theory work reported here arose from two experimental studies and one theoretical study. The $O(2^N)$ runtimes of the HT4 abductive inference engine motivated a review of the algorithm's performance. It was observed that the maximum number of goals reached by all worlds generated by HT4 was often similar to the number of goals reached in any single world. This observation, plus the stochastic reasoning work of Selman, Levesque, and Mitchell [24] prompted the following experiment in *random world generation*. HT4 collects together all the proof trees that connect inputs to goals, *then* sorts those proofs into worlds; i.e..maximal consistent subsets. That is, HT4 finds *all* worlds. In contrast, HT0 uses a *randomized world generation* strategy first [12]. Whenever a nondeterministic point is reached, HT0 *assumes* one option, picked at random. Future inferences must be consistent with this selected option. Occasionally, HT0 will retract all assumptions and restart. In this manner, some worlds will be generated, one at a time, each with a different set of randomly selected assumptions. The process continues until the number of reached goals by $Worlds_i$ is about the same as the maximum reached goals for $Worlds_1..Worlds_{i-1}$. In numerous experiments, HT4 was observed to run in time $O(2^N)$ while HT0 was observed to run in time $O(N^2)$. Further, for the problems which terminated with both HT0 and HT4, HT0 reached 98% of the goals found by HT4.

This result was unexpected (to say the least). HT0 is much simpler than HT4 and yet performs much better. More experiments were performed [18]. It was found that for millions of runs over tens of thousands of theories containing a varying number of artificially introduced errors, the result was the same: the difference in the number of goals reached by the full-worlds search of HT4 was only 6% different (on average) to the goals reached via the random world generation.

These empirical results were followed by a mathematical analysis of proof tree generation over an and-or graph containing contradictions [9, 10]. The analysis showed that the above empirical observations where not mere quirks of their domain. Rather, they reflect a basic feature of building proof trees through a space of inconsistencies. A random search will find some subset of the total number of possible proof trees within a theory that connects inputs to goals. The funnels used by these proofs may hence be different. A necessary, but not sufficient, condition for the optimism of funnel theory is that, when faced with a choice of funnels, random search will take the easy way out and use narrow funnels.

Mathematical simulations by Menzies, Cukic, Singh and Powell suggest that this is indeed the case [9, 10, 15]. The probability distribution of successfully building a proof tree from randomly selected inputs to a random term in a horn clause theory can be determined [15]. These distributions have pessimistic and optimistic bounds. For example, in the pessimistic case, 100 randomly selected input sets will miss over 95% of the terms in the theory. In the optimistic case, 100 randomly selected input sets will miss only 50% of the terms in the theory. If it is assumed that the goal required

a conjunction of all funnel terms, then a worst-case probability of reaching a term via a funnel can be computed. The optimistic and pessimistic distributions can be used to compute the probability of (1) reaching all the funnel terms, then (2) reaching the goal from the funnel. After computing these distributions for $wide \leftarrow \alpha * narrow$, the effects of increasing funnel size on funnel choice can be calculated.

Simulations of this mathematical model offer some interesting results. As might be expected, given two funnels of the same size (i.e. $\alpha \leftarrow 1$) then the probability of using one of them is 50%. As the difference in their size grows (i.e. α increases), then increasingly the narrower funnel is preferred to the wider funnel. The effect is quite pronounced. For example, for all the studied distributions, if a wider funnel is 2,3,4,5,6 times bigger than a narrow funnel, then in 60%,70%,80%,90%,95% (respectively) of the simulations, random search will be over 1,000,000 times as likely as to use the narrow funnel [10].

Narrower funnels imply fewer distinct behaviors since the cardinality of the funnels controls the range of behaviors in a theory containing contradictions. Narrow funnels explains the above empirical results. A random worlds search finds as many goals as the full worlds search since both searches were controlled by the same funnels. Further, the random worlds search runs faster since it can quickly sample the funnels without all the overheads of the more rigorous search.

4. TAR2

Assuming narrow funnels, then it is very simple to simulate and summarize RE theories. If the funnels are narrow, then HT0-style inference simulations of RE theories will quickly sample the range of behaviors in a system. Further, in the logs of those behaviors, the funnel attributes will appear very frequently since all pathways must use the same small set of attributes. Hence, assuming funnels, the treatment learning problem reduces to the task of finding high frequency attribute ranges that occur more often in the preferred behaviors than in the less desirable behaviors.

TAR2 is a treatment learning system that assumes (1) a partial ordering exists between classes and (2) datasets are generated from simulations of theories with narrow funnels. The goal of the learner is to find some minimal treatment strategy that improves the distribution of classes in the dataset. TAR2's input/output is shown in Figure 6. The terminology of that figure is explained below. Reserved variable names and types are shown in *italics*.

TAR2 will be explained using a case study based on the software capability maturity model from CMU's Software Engineering Institute (SEI-CMM) [21]). The chapter of [21] relating to CMM level 2 (informally defined software processes) was manually converted from text into a directed

Inputs:	*granularity*, *nchanges*	: postint;
	promising, *useful*	: posnum;
	classes	: associationArray;
	dataset	: set
	score(), *classify*()	: function
	now, *changes*	: select
Outputs:	*treatment*	: bandSelect

Figure 6: TAR2: inputs and outputs.

and-or graph. Each node represents some software management action. Each edge was augmented with numeric weights for the dollar cost of use some technique and the chances that this technique will contribute to the downstream node. These numerics are highly subjective and are drawn from $cost \in \{10, 20, 30, 40, 50\}$ and $chances \in \{0.1, 0.3, 0.9\}$. The resulting system had 124 control variables (all the costs and chances) and was executed 1000 times using the HT0-style inference engine described in [11]. This engine defines combination rules for costs and chances across an and-or network (for details, see [11]).

Since our cost and chances were so subjective, our case study tested if a ±30% change in any of these control variables had a significant impact in the total cost and chances of producing a good software project (as defined by SEI-CMM level 2). Hence, every numeric edge weight was converted to a random variable with a range $0.7X \leq X \leq 1.3X$.

All the classes known to TAR2 are stored in the *classes* association array which, in BNF[2] appears as:

$$classes ::== \{index\ class\}+$$

where *classes*[*i*] is "significantly worse" than *classes*[*j*] ($i < j$). In our case study, the classes were:

$$classes \leftarrow < (0, hilo), (1, lolo), (2, hihi), (3, lohi) >$$

denoting high/low, low/low, high/high, low/high (respectively) for the cost/chances of a good SEI-CMM level 2 project.

The user-supplied function "score()" operationalizes "significantly worse". We found an exponential scoring scheme works adequately such as $score(classificaions[i]) = 2^i$; i.e. low cost, high chance projects (lohi) are eight times better that high cost, low chances projects (hilo). A requirement of the *score*() function is that it must be able to compute one or more *best* classes; in this case $best \leftarrow lohi$.

A dataset was generated from the 1000 runs over our CMM level 2 theory. A *classify*() function was used to add a class to each item in the dataset.

$$\begin{aligned} dataset &::== \{\{attribute = value\}+\}+ \\ instances &::== \{\{attribute = value\} + class\}+ \end{aligned}$$

All values in the current implementation of TAR2 must be numeric so booleans are expressed as 0,1. In that implementation, -1 represents unknown. TAR2 descretized these numeric values by sorting all values for one attribute, then dividing them into a number of percentile *bands*. The number of bands is preset by the *granularity* variable. Note that as *granularity* increases, TAR2 explores the impact of more combinations of finer-grain ranges. By default, $granularity \leftarrow 3$.

It is recommended that *classify*() be designed such that instances of the "best classes" are nearly as common as the other classes. Our case study showed adequate performance using a *classify*() that generated the distribution seen in Figure 7A. This untreated distribution is called the *baseline distribution*. The baseline's *worth* was computed by summing relative frequencies of each class, weighted by the

[2]In this BNF notation, $W ::== XY|Z$ denotes that the structure W can contain the structures X, Y or Z. Also, $\{X\}+$ denotes one or more occurrences of the structure$X$.

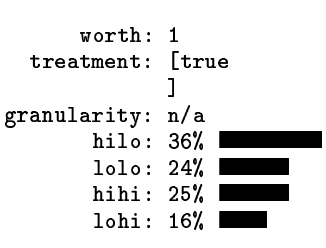


Figure 7A: the untreated baseline distribution.

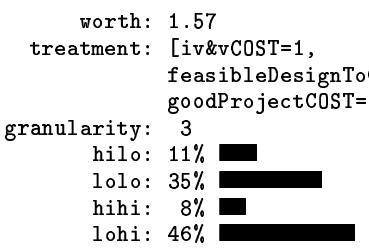


Figure 7B: after some treatment.

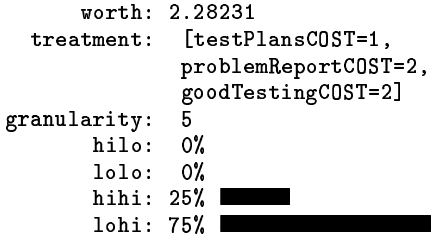


Figure 7C: after much treatment.

Figure 7: Improvements in distributions of classes generated by TAR2. At `granularity=3` (e.g. Figure 7A, Figure 7B), a treatment of `X=1` means reduce `X` to the lowest one-third percentile band. At `granularity=5` (e.g. Figure 7C), a treatment of `X=1` or `X=2` means reduce `X` to the lowest or second lowest (respectively) one-fifth percentile band.

$score()$ of that class; i.e.

$$worth \leftarrow \sum_{i \in classes} \left(\frac{score(i) * |instances\ with\ classes[i]|}{|instances|} \right)$$

The histogram of Figure 7A displays this baseline's *worth* as "1" since TAR2 reports the worth of other distributions as ratios of this baseline.

From a business perspective, the baseline distribution looks bad since it is highly skewed towards the worse classes. The goal of TAR2 is to find some *treatment* that improves on this baseline; e.g. change Figure 7A to Figure 7C. In the general case, a *treatment* is a *select* statement that restricts the ranges of an attribute:

$$\begin{aligned} select &::== true | \{attribute\ \{comp\}+\}+ \\ comp &::== op\ value \\ op &::== =< | < | = | > | >= \end{aligned}$$

A *comp* set bigger than one is a disjunction; e.g. age=2 or age=3. Otherwise, all the restrictions on attributes are "and-ed" together. The operation $SET \wedge SELECT$ extracts the subset of SET that satisfy $SELECT$.

TAR2 accepts as input two *select* statements: *now* and *changes*. *Now* represents the current state of the domain. Ideally, $now \leftarrow true$; i.e. the dataset was generated using only current conditions in the domain. However, it is not uncommon that users are accessing some historical dataset that may contain items not relevant to current conditions. The irrelevant parts are culled by *now*. *Changes* represents some desired zone within the dataset that the user wishes to approach. The attributes of the *changes* set are the control variables; i.e. the things that the user is willing to change in order to improve current conditions. The ranges within *changes* specify exactly what changes the user is considering.

In our case study, we sought any changes that make a significant difference to the *worth* of the dataset; i.e. $now \leftarrow true$ and *changes* contains all $A > -1$ for all attributes A; i.e. any change to any known band of an attribute was acceptable. In the current implementation, TAR2's *treatments* are limited to equality selects on percentile bands and no disjunctions; i.e.

$$\begin{aligned} bandSelect &::== \{attribute = percentileBand\}+ \\ percentileBand &:== 1..granularity \end{aligned}$$

The restriction of TAR2's *treatment* operators to "=" simplifies the implementation: everything internally can be handled as descrete sets. However, it does means that TAR2 cannot learn a continuous *treatment* such as $X > 1$.

The set

$$relevant \leftarrow instances \wedge (now \vee changes)$$

contains all instances in the current or desired situation. TAR2 searches *relevant* for attribute bands condoned by the *changes* set that have very different frequency in the best than in the non-best classes. *Relevant* may require much less memory than *instances*. Firstly, since we are only searching for treatments amongst the attributes mentioned in *changes*, *relevant* only needs to store the attributes found in the *change* set. Secondly, since descretization clumps together different values, many entries in *instances* could map to the same *relevant* entry. *Relevant* therefore contains a counter showing how many *instances* map into this *relevant* entry:

$$relevant ::== \{bandSelect\ class\ repeats\}+$$

Treatments are constructed by exploring subsets of the attribute bands seen in *relevant*. An interesting attribute band is one with a "sufficiently different" distribution in the best class than in the other classes. The input parameter *promising* represents our threshold for "sufficiently different". Let $a_x = b_y$ denote one attribute band within *changes* and

$$bestf \leftarrow \sum_{r \in relevantBest} r.repeats$$

denote the number of times $a_x = b_y$ occurs in the best classes where

$$relevantBest \leftarrow (relevent \wedge a_x = b_y \wedge class = best)$$

Let

$$xf \leftarrow \sum_{r \in relevantX} r.repeats$$

denote the number of times $a_x = b_y$ occurs in some other class x where

$$relevantX \leftarrow (relevent \wedge a_x = b_y \wedge class = x)$$

The value

$$deltaf \leftarrow (score(best) - score(x)) * \frac{xf}{bestf}$$

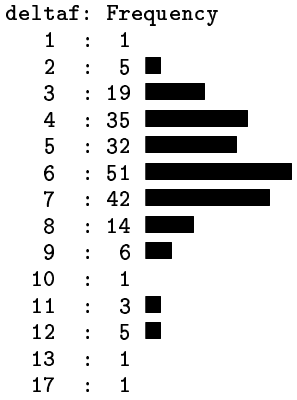


Figure 8: Distribution of $rounded(deltaf)$.

denotes the ratio of these two frequencies, weighted by the difference in the score of these two classes. The $deltafs$ seen in the CMM-2 *instances* are shown in Figure 8. $Treatments$ are formed from subsets of all bands $a_x = b_y$ with

$$deltaf \geq promising$$

where *promising* is a user-supplied variable. Note that *promising* increases, the size of *candidates* decreases. Referring to Figure 8, at *promising*←1, all 216 bands are *candidates* but at *promising*←12 there are only 7 *candidates* (the last 3 rows of the histogram).

The bulk of the time within TAR2 is spent exploring subsets of the *candidates*. Two parameters constrain this process. Firstly, only *treatments* of size *nchanges* (a user-supplied parameter) are explored. Since we seek to minimize the number of treatments, *nchanges* should be initialized to a low value (e.g. one) and only increased if the best treatments found so far are inadequate. Secondly, we only report treatments that have a $worth > useful$ where $useful$ is slightly more than the most worthy treatment found so far. $Useful$ is initialized to 1.05; i.e. initially we seek at least a 5% improvement in the *worth* of a treatment.

5. USING TAR2

An analyst using TAR2 manipulates $< useful, nchanges, promising, granularity >$ to discover a minimal treatment that has the most effect on the *worth* of the system. This manual manipulations of $< useful, nchanges, promising, granularity >$ described above *could* be automated. However, recalling the introduction, an RE tool should not automate the options exploration process. If an analyst and a domain expert walk through this process together, then the domain expert can usefully refine their purpose. For example, the original purpose of this case study stated above was somewhat vague:

PURPOSE 1. *Test the subjective nature of the numerics via a ±30% perturbation to all numerics.*

Based on our experience with the dataset, this purpose was refined several times (see below).

Our case study took 10 minutes to execute. TAR2 was run several times with analysts musing over the results in between each run. Initially we sought at least a 5% improvement using treatments on a single attribute after descretizing the continuous variables into three percentile bands; i.e. $< useful$←$1.05, nchanges$←$1, promising$←$1, granularity$←$3 >$. The biggest improvement was small: $worth$←1.135. It was hence concluded that we needed to try treatments that used more attributes. However, at $nchanges > 1$, TAR2 timed out due to too many *candidates*.

The number of *candidates* can be reduced by increasing *promising*. This is an effective strategy if there exists a small number of high-impact attribute ranges within the instances. To test this, the next run used *promising*←8 to find *nchanges*←2 treatments. This second run found a best *worth* found of 1.32. Experimenting further, the next run used *promising*←12 to find *nchanges*←3 treatments. This third run found the best *worth* of 1.57 shown in Figure 7B.

Note the recommendation of $iv\&vCOST = 1$ in Figure 7B. This recommendation requires a significant reduction in the cost of *iv&v* (independent software verification and validation); i.e. into the the lowest one-third percentile band. This research is partially funded by an IV&V group and that group might prefer methods that improve a projects without a major reduction in resources allocated to IV&V. Hence, we refined the purpose of the case study:

PURPOSE 2. *Find a change to the numerics that leads to better projects, without a major reduction in the IV&V effort.*

After recognizing this purpose, our business sense suggested we should try to generalize this refinement across all the variables. An interesting question to ask would be "is there anything we can reduce just a little to greatly improve the project?". At *granularity*←3, such small reductions are hard to see. Hence, we increased the granularity and made two new definitions. With *granularity*←5, we say that a treatment of the form $X = 1$ is a *major reduction* since it requires a reduction into the lowest one-fifth percentile; i.e. the 0% to 20% band. We also say that treatment of the form $X = 2$ is a *minor reduction* since it requires a reduction into the second lowest one-fifth percentile; i.e. the 21% to 40% band. With these definitions, we refined our purpose again:

PURPOSE 3. *Find a change to the numerics that leads to better projects, without a* major reduction *in most of the attributes in the treatment*

At $< useful$←$2, nchanges$←$3, promising$←$12,$ $granularity$←$5 >$, the best worth seen was Figure 7C. In this final distribution, all the low chances projects were removed and the majority of the remaining projects were low cost. Note that Figure 7C satisfies PURPOSE2 since it makes no comment on the IV&V costs. Further, since most of the recommended treatments are minor, Figure 7C also satisfies PURPOSE3.

6. DISCUSSION & EVALUATION

Case studies like the above example are useful under three conditions. Firstly, the managers of software projects must be able to implement the treatments proposed in Figure 7C. This is a domain-specific issue. By adjusting the *changes* select statement, TAR2 users can constrain the treatment generation process to just those attributes which can be changed.

Secondly, the treatment size must be a small subset of the possible values. Without this second condition, TAR2

users still need to debate numerous points. Where this second condition is satisfied, TAR2 is useful for RE since it focuses the stakeholders away from numerous low impact issues towards a small set of highly-critical issues. For example, in the above example, the space of 124 actions in our SEI-CMM level 2 model has been reduced to the three shown in Figure 7C.

Thirdly, the TAR2 algorithm must terminate. Most of TAR2's code is simple linear-time frequency counts of the attribute ranges in the *instances*. However the search through the subsets of the *candidates* is clearly exponential on the size of the *candidates*. TAR2 is only tractable if there exists a small number of attribute ranges with a large impact on the overall system; i.e. in domains with narrow funnels. The evidence for narrow funnels in §3 is encouraging, but hardly conclusive. However, looking further in the literature, we have much anecdotal evidence that domains often contain a small number of variables that are crucial in determining the behavior of the whole system. Concepts analogous to funnels have been called a variety of names including *master-variables* in scheduling [3]; *prime-implicants* in model-based diagnosis [23]; *backbones* in satisfiability [25]; *dominance filtering* in design [7]; and *minimal environments* in the ATMS [4].

Traditional methods of finding funnel-like variables can be complex; e.g. computing the minimal environments in the ATMS or the highest controversial assumptions of HT4 takes exponential time [17]. The core intuition of this paper was that such a search is a unnecessary. Since the funnels control the search space, *any* randomly selected pathway from inputs to goals must pass through the funnel (by definition). Hence, very simple tools like HT0 will suffice and the complexities of alternate truth-maintenance technologies (e.g. the ITMS [20]) is not required. In domains with master-variables of small sets of prime-implicants or small backbones or highly dominated solutions or small minimal environments, then (1) solutions generated via random search that cover goals will contain funnel variables with a high frequency; (2) simple treatment learners like TAR2 should work. One simple test for narrow funnels is to apply HT0 to an RE theory then try to find treatments using TAR2. If effective treatments can't be found, then narrow funnels are unlikely.

Random world generation with HT0 and treatment learning with TAR2 reduces, but does not resolve or remove, the arguments seen during RE. This is quite deliberate. Debates during RE will not and should not go away. Such debates can generates heat and light and insights into a domain that may remain hidden otherwise. Stakeholders must be free to argue about anything they like. But in a resource-bounded situation (e.g. any software development process), we can argue less by sorting the arguments according to their information gain. Our claim here is that these "most informative arguments" will be the ones relating to the minimal treatments.

It would be inappropriate to view TAR2 as a replacement for general learning tools like C4.5. The utility of TAR2 for tasks other than treatment learning from datasets generated from theories with narrow funnels using randomized world generation is an open research issue.

It would also be inappropriate to view TAR2's recommendations as perfect guarantees of better performance. RE theories are developed early in the software product life cycle and are therefore somewhat imprecise. Hence treatments learnt from RE theories can never be certain strategies. However, the generation of such treatments meets many of the requirements listed in the introduction for an RE tool. Treatments contain control actions that change the overall behavior of an RE tool. Hence the key disputes are the ones that effect which treatment is chosen. To assess the impact of different trade-offs, we only need to look for where those trade-offs appear in the treatments. If the trade-offs are not found in the treatments then they have little overall impact on the system and need not be discussed. Lastly, treatment learning lets us evaluate an RE theory: if the RE theory cannot generate acceptable projects then it needs to be changed.

Acknowledgements

Partial funding for this work was provided through the NASA IV&V facility by the NASA Office of Safety and Mission Assurance. This work was only possible due to the enthusiasm of Martin Feather, JPL, for exploring machine learning for requirements engineering.

7. REFERENCES

[1] I. Bratko, I. Mozetic, and N. Lavrac. *KARDIO: a Study in Deep and Qualitative Knowledge for Expert Systems*. MIT Press, 1989.

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

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

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

[5] M. Feather, H. In, J. Kiper, J. Kurtz, and T. Menzies. First contract: Better, earlier decisions for software projects. In *Submitted to the ACM CIKM 2001: the Tenth International Conference on Information and Knowledge Management*, 2001. Available from `http://tim.menzies.com/pdf/01first.pdf`.

[6] 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.

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

[8] R. Madachy. Heuristic risk assessment using cost factors. *IEEE Software*, 14(3):51–59, May 1997.

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

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

[11] T. Menzies and J.D. Kiper. How to argue less. In *Submitted to the ACM CIKM 2001: the Tenth International Conference on Information and*

Knowledge Management, 2001. Available from `http://tim.menzies.com/pdf/01jane.pdf`.

[12] T. Menzies and C.C. Michael. Fewer slices of pie: Optimising mutation testing via abduction. In *SEKE '99, June 17-19, Kaiserslautern, Germany. Available from `http://tim.menzies.com/pdf/99seke.pdf`*, 1999.

[13] T. Menzies, J. Powell, and M. E. Houle. Fast formal analysis of requirements via 'topoi diagrams'. In *ICSE 2001*, 2001. Available from `http://tim.menzies.com/pdf/00fastre.pdf`.

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

[15] Tim Menzies, Bojan Cukic, Harhsinder Singh, and John Powell. Testing nondeterminate systems. In *ISSRE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00issre.pdf`.

[16] T.J. Menzies. *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales, 1995. Available from `http://tim.menzies.com/pdf/95thesis.pdf`.

[17] 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://tim.menzies.com/pdf/96aim.pdf`.

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

[19] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions of Software Engineering*, 18(6):483–497, June 1992.

[20] P. Pandurang Nayak and Brian C. Williams. Fast context switching in real-time propositional reasoning. In *Proceedings of AAAI-97*, 1997. Available from `http://ack.arc.nasa.gov:80/ic/projects/mba/papers/aaai97.ps`.

[21] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber. Capability maturity model, version 1.1. *IEEE Software*, 10(4):18–27, July 1993.

[22] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

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

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

[25] Josh Singer, Ian P. Gent, and Alan Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.

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