# Learning Minimal Policies for Software Projects

TIM MENZIES, West Virginia University, WV, USA
and
JAMES KIPER, Miami University, Ohio, USA
and
JEREMY GREENWALD, MathWorks, USA
and
YING HU, Vancouver, BC, Canada
and
DAVID RAFFO, and and SIRI-ON SETAMANIT , Portland State University, OR, USA

---

Blab blah

Categories and Subject Descriptors: aa [**bb**]: cc

General Terms: dd

Additional Key Words and Phrases: ee

---

Dr. Menzies is with the Lane Department of Computer Science, West Virginia University and can be r eached at `tim@menzies.us`,
Dr. Kiper is with the Computer Science and Systems Analysis Department, School of Engineering & Applied Science, Miami University, and can be reached at `kiperjd@muohio.edu`.
Ms. Hu is a software designer in Vancouver, British Columbia and can be reached at `huying_ca.@yahoo.com`.
Mr. Greenwald, formerly with Computer Science at Portland State University, now works at MATHWORKS `jegreen@cecs.pdx.edu`.
Dr. Raffo is a Associate Professor in the School of Business Administration, Portland State University, and can be reached at `raffod@sba.pdx.edu`.
Ms. Setamanit is a graduate student at Portland State Universit. and can be reached at `sirion@pdx.edu`.
The research described in this paper was carried out at Miami University, West Virginia University and Portland State University under contracts and sub-contracts with NASA's Software Assurance Research Program. Reference herein to any specific commercial product, process, or service by trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.
Download an earlier draft from `http://menzies.us/pdf/07succinct.pdf`.

## 1. INTRODUCTION

Consider a software manager using a software process model to control their project. Exploring all the "what-ifs" within those models can be a daunting tasks. For example, 20 binary decisions require, in the worst case, $2^{20} > 1,000,000$ what-ifs. For another example, after interviewing the managers of three NASA systems regarding their process options, we found that they were considering up to $10^9$ options[1].

In the $21^{st}$ century, it is possible to use standard desktop machines to explore a very large option space. Many businesses under-utilize their CPUs once staff go home for the day. Those machines can be combined into a CPU farm which. If the process models are not database or GUI intensive, then such a CPU farm can can work through billions of options in an overnight run.

However, once all those options are created, they must be summarized. That is, it is not enough to *generate* a multitude of alternatives, we must also *select* the subset of the possible options that matter then most. This paper discusses a summarization technology that exploits two naturally occuring properties of many models: *clumps* and *collars*

The number of configuration possibilities within a model can be dauntingly large. A model with 20 binary choices has $2^{20} > 1,000,000$ possible configurations, far beyond the capability of human comprehension. Standard model comprehension methods such as design of experiments have limited utility when the model is non-continuous, very large, or contains noise.

Hence, since 2000, we have been exploring model comprehension using data miners. After randomly sampling those configurations, running the resulting model, scoring the output with some oracle, our data miners are used to configure options that most improve model output [**?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**]. This paper synthesizes that prior work and presents new case studies. Originally, our work was inspired by Bratko's combination of qualitative modeling and data mining [**?**]. We found that standard data miners may be inadequate for exploring model configuration possibilities since they often yield results that are incomprehensible to humans:

—Neural nets never generate a succinct generalization of their knowledge [**?**];

—The random search of genetic algorithms can produce models that are too complex to understand [**?**];

—Most decision/regression tree learners such as C4.5 [**?**] or CART [**?**] execute in local top-down search, with no memory between different branches. Hence, the same concept can be needlessly repeated many times within the tree. Consequently, such trees can be cumbersome, needlessly large, and difficult to understand.

The detailed, complex and arcane output generated by standard data miners is often superfluous. Firstly, many human experts can not (or will not) read complex theories learned by a data miner. Secondly, numerous empirical and theoretical

[1] A COCOMO model with 12 unknowns was being analyzed; each unknown has up to 6 possible values; $6^{12} > 1,000,000,000$.

results argue that, in the usual case, models are controlled by a handful of of key variables. If data miners are restricted to just returning models containing those keys, then the learned model will be very small indeed.

Thus, we propose a different kind of data miner to assist analysts in exploring the configuration possibilities within their models. Traditional machine learners like C4.5 generate classifiers that assign a class symbol to an example. Our preferred method, called *treatment learning* generates just the *differences* in the key variables among different outcomes.

The rest of this paper is structured as follows. The twin topics of model-based SE and search-based SE are introduced. A special case of *hard models* are discussed that are difficult to manage using standard methods. Evidence is then presented that, often, the key variables that control a model are few in number. The problem of hard models therefore can be reduced to just the problem of understanding the key variables. This will motivate the design of the TAR2 *treatment learner*. Case studies will then be presented showing that, in many domains, TAR2 finds a small number of controlling variables. Such succinct controllers have many advantages:

—Smaller models are easier to explain (or audit).

—Miller has shown that models generally containing fewer variables have less variance in their outputs [**?**].

—The smaller the model, the fewer are the demands on interfaces (sensors and actuators) to the external environment. Hence, systems designed around small models are easier to use (less to do) and cheaper to build.

In short, for a certain class of *hard modeling* problems, the TAR2 treatment learner can dramatically simplify the task of understanding the configuration possibilities within model-based SE. Hence, our conclusion will be that it is useful to augment standard modeling methods with treatment learning. As to other modeling problems, TAR2 is *not* indicated for low-dimensionality linear continuous models built in domains that have no noise or other uncertainties, and where managers have full control over all model inputs.

## 2. RELATED WORK

Traditional methods may not yield succinct control policies from software process models. One such traditonal method is *binary plotting*; i.e. $N$ model inputs are plotted against some output score in $N$ two-dimensional binary plots. The best constraints on the inputs are those that select for the best output. However, if noise in a models yields plots that are not smooth, then it becomes a time-consuming and subjective task to examine $N$ binary plots to detect regions of input parameters that most improves the outputs. Also, binary plotting will not find *conjunctions* of input constraints that contribute to better output. Further, as the dimensionality of model input increases, the cognitive effort required to study the plots can become prohibitively expensive.

Many researchers have developed impressive visual environments for decreasing the cognitive overload associated with exploring a multi-dimensional space. Such visualizations help analysts explore visual information, but they present their own challenges. Nevertheless, there are still limits on how many dimensions can be

displayed (e.g. we have yet to see effective visualizations for more than a ten dimensionality space). As the visualization environment grows more sophisticated, some users find they have traded a data browsing problem with the new problem of exploring the full range of the effects of all the controls.

When manual exploring of options fail, automatic methods can be applied. Optimization packages can be applied to data or the equations of a system to find "sweet spots" that maximize the score resulting from model outputs. For example, a canonical *sensitivity analysis* method [**?**] might be to compute eigenvectors of a linear system in order to understand its long-term temporal behavior [**?**; **?**]. Alternatively, using design of experiments [**?**], we might exercises an existing model to shed light on the response surface of the model (DOE does identify gradients and key parameters for a model).

While useful for some models, these automatic methods do not apply to all models. For example, optimization methods can fail for non-linear models. Any model with an "if" statement introduces a "cliff" where the effects of inputs on outputs abruptly changes. For such models, there is no linear continuous solution that applies either side of the "cliff". Model uncertainty also complicates sensitivity analysis. For example, the eigenvector technique described would yield spurious results if the coefficients on the models are not known with certainty. Lastly, a DOE analysis can be complicated by the dimensionality, noise, and visualization problems described above.

When sensitivity experiments and DOE are not appropriate, other methods such the cross-entropy method [**?**]; simulated annealing [**?**], tabu search [**?**], or genetic algorithms [**?**] might be employed. All these methods have been used for search-based software engineering XXX

More generally, traditional methods often assume total knowledge and total control of the input space. This is often impractical. Consider, for example, Mintzberg [**?**] study of how real-world decisions are made. He found:

—56 U.S. foremen who averaged 583 activities in an eight-hour shift (one every 48 seconds).

—160 British managers who worked for half an hour or more without interruption only once every two days.

These empirical observations do not fit a traditional optimization model of decision making were all options are systematically studied, organized, co-ordinated and controlled. In *hard modeling problems*, human agents must make decisions using:

—limited time and computational ability;

—limited computational ability (or limited time for computation);

—limited knowledge about decision alternatives;

—uncertainty about possible outcomes of decisions

—uncertainty about pay-offs;

—no more than a partial ordering of preferences;

—limited information about probabilities of outcomes.

Herbert Simon [**?**] defined and explored such hard modeling problems using a data structure called *state space* [**?**]. In terms of model-based SE, state space is the set of

options and option selection operators within a model. Further, in hard modeling problems, large portions of the state space are uncertain. Simon argued that in such state spaces, searching for optimal solutions is a spurious goal. Rather, agents can only make just enough decisions that are just good enough. In Simon's terminology, such decisions are *satisficing*.

Our contribution to hard modeling is to comment that (1) *satisficing* solutions can be achieved by ignoring certain irrelevant or redundant details within a model; (2) surprisingly simple methods can find what irrelevancies to ignore. Treatment learners propose "treatments"; i.e. constraints on a small subset of the model inputs. The other inputs are left to vary at random. That is, apart from generating very succinct solutions, treatment learning offers solutions that are stable despite uncertainty in the non-treated variables.

## 3. COLLARS AND CLUMPS

This research assumes that many models can be controlled by a small number of key variables which we call *collars*. Collars restrict the behavior of a model such that their state space *clumps*; i.e. only small number of states are used at runtime. If so, the output of a data miner could be simplified to constrain just the collar variables that switch the system between a few clumps.

To visualize collars, imagine an execution trace spreading out across a program. Initially, the trace is very small and includes only the inputs. Later, the trace spreads wider as *upstream* variables effect more of the *downstream* variables (and the inputs are the most *upstream* variables of all). At some time after the appearance of the inputs, the variables hold a set of values. Some of these values were derived from the inputs while others may be default settings that, as yet, have not been affected by the inputs. The union of those values at time $t$ is called the *state* $s_t$ of the program at that time.

Multiple execution traces are generated when the program is run multiple times with different inputs. These traces reach different branches of the program. Those branches are selected by tests on conditionals at the root of each branch. The *controllers* of a program are the variables that have different settings at the roots of different branches in different traces. Programs have *collars* when a handful of the controllers in an early state $s_t$ control the settings of the majority of the variables seen in later states .

A related effect to *collars* is *clumping*. If a program has $v$ variables with range $r$, then the maximum number of states is $r^v$. Programs *clump* when most of those states are never *used* at runtime; i.e. $|used|/\left(r^v\right) \approx 0$. Clumps can cause collars:

—The size of *used* is the cardinality cross product of the ranges seen in the controllers.

—If that cardinality is large, many states will be generated and programs won't clump.

—But if that cross product is small, then the deltas between the states will be small – in which case controlling a small number of collar variables would suffice for selecting what states are reached at runtime.

There is much theoretical and empirical evidence for expecting that many models often contain *collars* and *clumps*.

### 3.1 Theoretical Evidence

With Singh [**?**], we have shown that collars are an expected property of Monte Carlo simulations where the output has been discretized into a small number of output classes. After such a discretization, many of the inputs would reach the same goal state, albeit by different routes. The following diagram shows two possible distinct execution paths within a Monte Carlo simulation both leading to the same goal; i.e. $a \rightarrow goal$ or $b \rightarrow goal$.

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

Each of the terms in lower case in the above diagram represent a probability of some event; i.e. $0 \leq \{a_i, b_i, c, d, goal_i\} \leq 1$. For the two pathways to reach the *goal*, they must satisfy the collar $M$ or the larger collar $N$ (each collar is a conjunction). As the size of $N$ grows, the product $\prod_{j=1}^{N} b_j$ decreases and it becomes less likely that a random Monte Carlo simulation will take steps of the larger collar $N$.

The magnitude of this effect is quite remarkable. Under a variety of conditions, the narrower collar is thousands to millions of times more likely. For example, when $|M| = 2$ and $N > M$, the condition for selecting the larger collar is $\frac{d}{c} \geq 64$; i.e. the larger collar $N$ will be used only when the $d$ pathway is dozens of times more likely than $c$. The effect is more pronounced as $|M|$ grows; at $|M| = 3$ and $N > M$, the condition is $\frac{d}{c} \geq 1728$; i.e. to select the larger collar $N$, the $d$ pathway must be thousands of times more likely than $c$ (for more details, see [**?**]). That is, when the output space is discretized into a small number of outputs, and there are multiple ways to get to the same output, then a randomized simulation (e.g. a Monte Carlo simulation) will naturally select for small collars.

While the mathematics may be arcane, the intuition is simple. Suppose all the power goes out in a street of hotels. If all those hotels were designed by different architects then their internal search spaces would be different (number of rooms per floor, distance from each room to a flight of stairs, number of stairwells, etc). After half an hour, some of the guests bumble around in the dark and get outside to the street. Amongst the guests that have reached the street, there would be more guests from hotels with simpler internal search spaces (fewer rooms per floor, less distance from each room to the stairs, more stairwells).

As to *clumping*, Druzdel [**?**] observed this effect in a medical monitoring system. The system had 525,312 possible internal states. However, at runtime, very few were ever reached. In fact, the system remained in one state 52% of the time, and a mere 49 states were used, 91% percent of the time. Druzdel showed mathematically that there is nothing unusual about his application. If a model has $n$ variables, each with its own assignment probability distribution of $p_i$, then the probability that the model will fall into a particular state is $p = p_1 p_2 p_3 ... p_n = \prod_{i=1}^{n} p_i$. By taking logs

of both sides, this equation becomes

$$\ln p = \ln \prod_{i=1}^{n} p_i = \sum_{i=1}^{n} \ln p_i \quad (1)$$

The asymptotic behavior of such a sum of random variables is addressed by the central limit theorem. In the case where we know very little about a model, $p_i$ is uniform and many states are possible. However, the *more* we know about the model, the *less* likely it is that the distributions are uniform. Given enough variance in the individual priors and conditional probabilities or $p_i$, the expected case is that the frequency with which we reach states will exhibit a log-normal distribution; i.e. a small fraction of states can be expected to cover a large portion of the total probability space; and the remaining states have practically negligible probability.

The assertion that many types of models display this clumping behavior is quite important for the style of data mining (treatment learning) that we advocate. In application to a clumping model with collars, Monte Carlo simulation, followed by TAR2, suffices to summarize that model in an effective way:

—TAR2's rules never need to be bigger than the collars. Hence, if the collars are small, TAR2's rules can also be small.

—If a model clumps, then, very quickly, a Monte Carlo simulation would sample most of the reachable states. TAR2's summarization of that simulation would then include most of the important details of a model.

### 3.2 Empirical Evidence

Empirical evidence for clumps first appeared in the 1950s. Writing in 1959, Samuel studied machine learning for the game of checkers [**?**]. At the heart of his program was a 32-term polynomial that scored different configurations. For example, *king center control* means that a king occupies one of the center positions. The program learned weights for these variable coefficients. After 42 games, the program had learned that 12 variables were important, although only 5 of these were of any real significance.

Decades later, we can assert that deleting irrelevant variable has proven to be a useful strategy in many domains. For example, Kohavi and John report experiments on 8 real world datasets where, on average, 81% of the non-collar variables can be ignored without degrading the performance of a model automatically learned from the data [**?**].

If models contain collars, or if the internal state space clumps, then much of the reachable parts of a program can be reached very quickly. This *early coverage* effect has been observed many times. In a telecommunications application, Avritzer, Ros, & Weyuker found that a sample of 6% of all inputs to this system covered 99% of all inputs seen in about one year of operation (and a sample of just over 12% covered 99.9%) [**?**]. Further evidence for early coverage can be found in the mutation testing literature. In mutation testing, some part of a program is replaced with a syntactically valid, but randomly selected, variant (e.g. switching "less than" signs to "greater than"). This method of testing is useful for getting an estimate of what percentage of errors have been discovered by testing. Wong compared results using X% of a library of mutators, randomly selected (X $\in$ {10,15,... 40,100}). Most of

what could be learned from the program could be learned 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 [**?**]. The same observation has been made elsewhere by Budd [**?**] and Acree [**?**].

If the space of possible execution pathways within a program are limited, then program execution would be observed to clump since it could only ever repeat a few simple patterns. Empirically such limitations have been observed in procedural and declarative systems. Bieman and Schultz [**?**] report that 10 or fewer paths through programs explored 83% of the du-pathways (a du-path is a set of statements in a computer program from a definition to a use of a variable. This is one common form of structural coverage testing.) Harrold [**?**] studied the control graphs of 4000 Fortran routines and 3147 C functions. Potentially, the size of a control graph may grow as the square of the number of statements (in the case where every statement was linked to every other statement.) This research found that, in these case studies, the size of the control graph is a linear function of the number of statements. In an analogous result, Pelánek reviewed the structures of dozens of formal models and concluded that the internal structure of those models was remarkably simple: "state spaces are usually sparse, without hubs, with one large SCC [strongly connected component], with small diameter [2] and small SCC quotient"[3] [**?**]. This sparseness of state spaces was observed previously by Holtzmann where he estimate the average degree of a vertex in a state space to be 2 [**?**].

Pelánek hypotheses that these "observed properties of state spaces are not the result of the way state spaces are generated nor of some features of specification languages but rather of the way humans design/model systems" [**?**]. Pelánek does not expand on this, but we assert that generally SE models are simple enough to be controlled by treatment learning since they were written by humans with limited short-term memories [**?**] who have difficulty designing overly-complex models.

## 4. DATA MINING WITH COLLARS AND CLUMPS

The TAR2 *treatment learner* [**?**] is a data miner that is specialized for generating models containing only collar variables. TAR2 finds the difference between classes. Formally, the algorithm is a *contrast set learner* [**?**; **?**] that uses *weighted classes* [**?**] to steer the inference towards the preferred behavior. We call TAR2's output "treatments" since the minimal rules generated by the algorithm are similar to medical treatment policies that try to achieve the most benefit, with the least intervention. The core intuition of TAR2 is that it is unnecessary to search for the collars– they will reveal themselves after some limited random sampling. To see that, recall that collar variables control the settings in the rest of the system. Any execution trace that reaches a goal must pass through the collars (by definition). Therefore, to find the collars, all an algorithm needs to do is find the attribute ranges with very different frequencies in traces that reach different goals.

Detecting collars via this sampling method is very simple to implement. Consider a log of golf playing behavior shown in Figure 0**??**. This log contains four attributes

[2]The diameter of a graph (of a state space here) is the number of edges on the largest shortest path between any two vertices.
[3]SCC quotient is a measure of the complexity of a graph.

(outlook, temperature, humidity, wind) and 3 classes (none, some, lots) that convey the amount of golf played. We recommend an exponential scoring system for the classes, starting at two[4]. For example, our golfer could weight the classes in Figure 0?? as *none=2* (worst), *some=4*, *lots=8* (best).

TAR2 seeks attribute ranges that occur frequently in the highly weighted classes and rare in the lower weighted classes. Let $a.r$ be some attribute range e.g. *outlook.overcast* means that the outlook is for overcast skies. $\Delta_{a.r}$ is a heuristic measure of the worth of $a.r$ to improve the frequency of the *best* class. $\Delta_{a.r}$ uses the following definitions:

$X(a.r)$:. is the number of occurrences of that attribute range in class $X$; e.g. in this data *lots(outlook.sunny)=2* since there are 2 cases with outlook = *sunny* and class = *lots*.

$all(a.r)$:. is the total number of occurrences of that attribute range in all classes; e.g. *all(outlook.sunny)=5*.

*best*:. the highest scoring class; e.g. $best = lots$;

*rest*:. the non-best class; e.g. $rest = \{none, some\}$;

*weight*:. The weight of a class $X$ is symbolized by $\$X$;. (Thus, $\$best = 8$.)

$\Delta_{a.r}$ is calculated as follows:

$$\Delta_{a.r} = \frac{\sum_{X \in rest} (\$best - \$X) * (best(a.r) - X(a.r))}{all(a.r)}$$

When $a.r$ is *outlook.overcast*, then $\Delta_{outlook.overcast}$ is calculated as follows:

$$\frac{\overbrace{((8-2)*(4-0))}^{lots \rightarrow none} + \overbrace{((8-4)*(4-0))}^{lots \rightarrow some}}{4+0+0} = \frac{40}{4} = 10$$

To *build* a treatment, TAR2 explores combinations of attribute ranges up to some user-specified maximum size $s$ (where the size $s$ is the number of attribute ranges in a conjunction of attributes). Given $n$ attributes, the size of this search is $\frac{n!}{s!(n-s)!}$. To make this search feasible, TAR2 must keep $s$ small. Therefore, TAR2 first assesses each attribute range, in isolation, i.e., with s = 1. A preliminary pass builds one singleton treatment for each attribute range. The attribute ranges are then scored by the $\Delta$ of these singleton treatments. Treatment generation is constrained to just the attribute ranges with a score greater than a user-supplied threshold (default value= 1; maximum useful value yet found= 7).

To *apply* a treatment, TAR2 rejects all example entries that contradict the conjunction of the attribute ranges in the treatment. E.g., if the treatment was $humidity \geq 85 \wedge windy = true$, then 11 of the lines of Figure 0?? would be rejected. The ratio of classes in the remaining examples is compared to the ratio of classes in the original example set (in the humidity and wind treatment just given, this ratio would be 3/14). The *best treatment* is the one that most increases the relative percentage of preferred classes. In our golf example, a single best treatment

[4]If the weights run, say, {bad=0,ok=1,good=2} then the difference from *bad* to *ok* scores the same as *ok* to *good*. An exponentially weighting scheme, starting at two, finds greater and greater rewards moving to better classes. For further details, see [?].

| *outlook* | *temp(°F)* | *humidity* | *windy?* | *class* | *weight* |
|---|---|---|---|---|---|
| *sunny* | *85* | *86* | *false* | *none* | *2* |
| *sunny* | *80* | *90* | *true* | *none* | *2* |
| *sunny* | *72* | *95* | *false* | *none* | *2* |
| *rain* | *65* | *70* | *true* | *none* | *2* |
| *rain* | *71* | *96* | *true* | *none* | *2* |
| *rain* | *70* | *96* | *false* | *some* | *4* |
| *rain* | *68* | *80* | *false* | *some* | *4* |
| *rain* | *75* | *80* | *false* | *some* | *4* |
| *sunny* | *69* | *70* | *false* | *lots* | *8* |
| *sunny* | *75* | *70* | *true* | *lots* | *8* |
| *overcast* | *83* | *88* | *false* | *lots* | *8* |
| *overcast* | *64* | *65* | *true* | *lots* | *8* |
| *overcast* | *72* | *90* | *true* | *lots* | *8* |
| *overcast* | *81* | *75* | *false* | *lots* | *8* |

Fig. 1. A log of some golf-playing behavior.

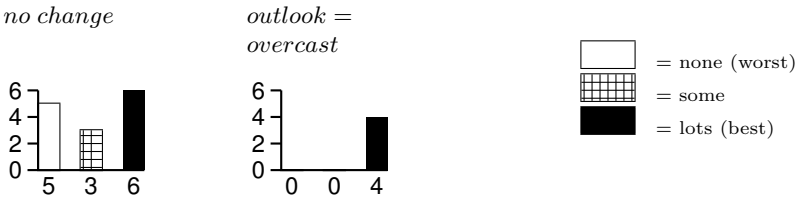


Fig. 2. Finding treatments that can improve golf playing behavior. With no treatments, we only play lots of golf in $\frac{6}{5+3+6} = 57\%$ of cases. However, assuming *outlook=overcast*, we play golf lots of times in 100% of cases.

| | | # columns | | | |
|---|---|---|---|---|---|
| domain | # rows | # numeric | # discrete | #class | time(sec) |
| iris | 150 | 4 | 0 | 3 | $< 1$ |
| wine | 178 | 13 | 0 | 3 | $< 1$ |
| car | 1,728 | 0 | 6 | 4 | $< 1$ |
| autompg | 398 | 6 | 1 | 4 | 1 |
| housing | 506 | 13 | 0 | 4 | 1 |
| pageblocks | 5,473 | 10 | 0 | 5 | 2 |
| cocomo | 30,000 | 0 | 23 | 4 | 2 |
| reacheness | 25,000 | 4 | 9 | 4 | 3 |
| circuit | 35,228 | 0 | 18 | 10 | 4 |
| reacheness2 | 250,000 | 4 | 9 | 4 | 23 |
| pilot | 30,000 | 0 | 99 | 9 | 86 |

Fig. 3. Runtimes for TAR2 on different domains. First 6 data sets come from the UC Irvine machine learning data repository [**?**]; "cocomo" comes from a COCOMO software cost estimation model [**?**]; "pilot" comes from the NASA Jet Propulsion Laboratory [**?**].

was generated containing *outlook=overcast*. Figure 0?? shows the class distribution before and after that treatment. That is, if we select a vacation location with *overcast* weather, then we should be playing *lots* of golf, all the time.

In practice, despite the $\frac{n!}{s!(n-s)!}$ search, TAR2 scales well. Figure 0?? shows TAR2's runtime on 11 data sets with varying numbers of rows and columns. Running on a relatively slow machine (a 333 MHz Windows machine with 512MB of ram), TAR2 terminated in tens of seconds, even on data sets with up to 250,000 rows, each with nearly 100 attributes.

## 5. CASE STUDIES

In theory, we expect that many models contain collars and clumps. If so, tools like TAR2 should be able to find tiny treatments that control the behavior of the models. This section tests that theory on several case studies.

### 5.1 Studying the CMM

The previous studied explored a numeric model where all the influences were precisely specified. This second case study takes a numeric model and adds a large degree of uncertainty in the numerics. This second study shows that, even in presence of large degrees of uncertainty, TAR2 can still find useful treatments.

An important feature of this second study is that it analyzes a class of models that can defeat standard methods. The model contains dozens of if-then rules; i.e. it is neither linear nor continuous: small changes in the environment can lead to "cliffs" where the model behavior changes abruptly. Also, the model contains non-deterministic choices (see the *rany* operator, discussed below) and so its behavior can be highly noisy.

This study uses a rule-based model of the costs and benefits model of CMM level 2 (hereafter, CMM2) [?, p125-191]. We elected to study CMM2 since, in our experience, many organizations can achieve at least this level. CMM2 is less concerned with issues of (e.g.) which design pattern to apply, than with what overall project structure should be implemented. Improving CMM2-style decisions is important since in early software life cycle, many CMM2-style decisions affect the resource allocation for the rest of the project.

CMM2 was encoded using the JANE propositional rule-based language [?]. JANE's rules take the form *Goal if SubGoals* such as the one shown in Figure 0??.

```
stableRequirements
    if   effectiveReviews
    and requirementsUsed
    and sEteamParticipatesInPlanning
    and documentedRequirements
    and sQAactivities
    and (reviewRequirementChanges
          rany softwareConfigurationManagement
          rany baselineChangesControlled
          rany workProductsIdentified
          rany softwareTracking
        ).
```

Fig. 4. Part of CMM2, encoded in the JANE language.

JANE is a backward chaining language: to prove a *Goal*, JANE tries to find rules that prove each of the *SubGoals*. Each *SubGoal* contributes some *Cost* and *Chances* to the *Goal*. JANE's *Chances* define the extent to which a belief in one vertex can propagate to another. *Cost*s let an analyst model the common situation where some of the *Cost* of some procedure is amortized by reusing its results many times. Hence, the *first* time we use a proposition, we incur its *Cost* but afterwards, that proposition is free of charge.

The *Cost* and *Chances* of a proposition are either provided by the JANE programmer or computed at runtime via a traversal of the rules:

—When searching *X if not A*, the *Chances* of *X* are *1-Chances(A)* and $Cost(X) = Cost(A)$.

—When searching *X if A and B and C*, the *Chances* and *Costs* of *X* are (respectively) the product of the chances and the sum of the costs of *A,B,C*.

—When searching *X if A or B or C*, then the *Cost* and *Chances* of *X* are taken from the first member of *A,B,C* that is satisfied.

These *and*, *or*, *not* operators can be insufficient to capture the decision making of business users. For examples, in our experience, business users select CMM2 options, often in a somewhat arbitrary manner. To model this, JANE includes a *rany* operator (short for "random any"):

—The *rany* operator is like *or* except that (e.g.) *X if A rany B rany C* succeeds if some random number of *A,B,C* (greater than one) succeeds. Unlike *and*, *or* which explore their operands in a left-to-right order, *rany* explores its *SubGoals* is a random order. If at least one succeeds, then the *Cost* and *Chances* of *X* is the sum and product (respectively) of the *Cost* and *Chances* of the satisfied members of *A,B,C*.

| | |
|---|---|
| *baselineAudits, baselineChangesControlled, changeRequestsHandled, changesCommunicated, configurationItemStatusRecorded, deviationsDocumented, documentedDevelopmentPlan, documentedProjectPlan, earlyPlanning, formalReviewsAtMilestones, goodUnitTesting, identifiedWorkProducts, periodicSoftwareReviews* | *planRevised, requirementsReview, requirementsUsed, reviewRequirementChanges, risksTracked, SCMplan, SCMplanUsed, SElifeCycleDefined, SEteamParticipatesInPlanning, SEteamParticipatesOnProposal, SQAauditsProducts, SQAplan, SQAplanUsed, SQAreviewActivities, workProductsIdentified* |

Fig. 5. Management actions in the CMM2 model. SQA= software quality assurance and SCM= software configuration management)

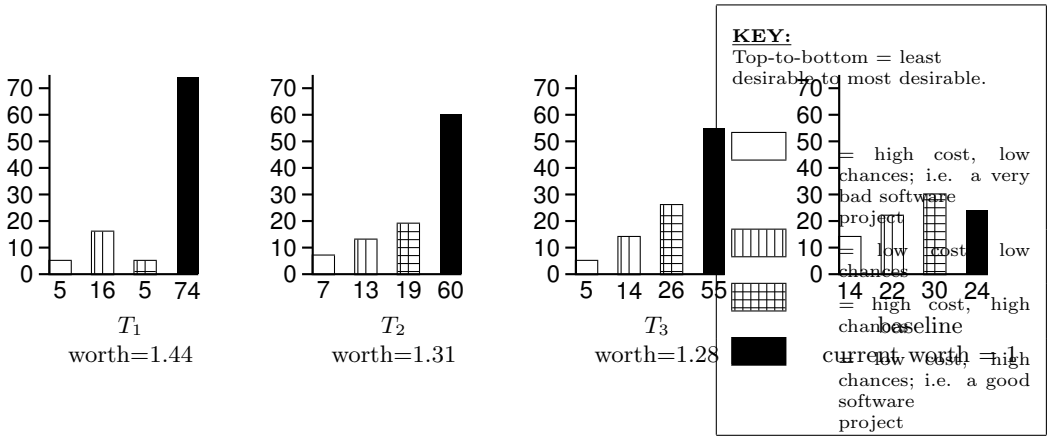


Fig. 6. Ratios of different software project types seen in four situations.

*Rany* is useful when searching for subsets that contribute to some conclusion. For example, the JANE rule in Figure 0**??** offers several essential features of *stableRequirements* plus several optional factors relating to monitoring change in evolving projects – the essential features are *and*-ed together while the optional factors are *rany*-ed together.

Figure 0**??** includes 11 propositions. Our model of CMM2, written in JANE, has 55 propositions ($range = \{t, f\}$). Of those 55 propositions, 27 were identified by our users as actions that could be changed by managers (see Figure 0**??**).

Apart from *rany*, JANE supports one other mechanisms for exploring the space of possibilities within CMM2. When defining *Cost*s and *Chances*, the programmer can supply a *range* and a *skew*. For example:

```
goodUnitTesting and cost = 1 to +5
```

defines the *cost* of *goodUnitTesting* as being somewhere in the range 1 to 5, with the mean skewed slightly towards 5 (denoted by the "+").

Similarly, while all the *Chances* values were based on expert judgment, their precise value is subjective. Hence, each such *Chances* value $X$ was altered to be a range

```
chances = 0.7*X to 1.3*X
```

During a simulation, the *first* time a *Cost* or *Chance* is accessed, it is assigned randomly according to the range and skew. The assignment is cached so that all subsequent accesses use the same randomly generated value. After each simulation, the cache is cleared. After thousands of simulations, JANE can sample the "what-if" behavior resulting from different assignments within the range and many different *rany* choices.

Data from 2000 simulations was passed from the CMM2 model to TAR2. Each simulation was classified into one of four classes:

—*class=0*: High cost, low chance;
—*class=1*: Low cost, low chance;

$T_1$:. *requirementsUsed.Cost=lower and*
*not periodicSoftware-Reviews and*
*formalReviewsAtMilestones.Cost=lower*
$T_2$:. *requirementsUsed.Cost=lower and*
*goodUnitTesting.Cost=middle and*
*formalReviewsAtMilestones.Cost=lower*
$T_3$:. *goodUnitTesting.Cost=lower and*
*periodicSoftwareReviews.Cost=middle and*
*formalReviewsAtMilestones.Cost=lower*

Fig. 7. The three best treatments found in the CMM2 model.

—*class=2*: High cost, high chance;
—*class=3*: Low cost, high chance.

That is, our preferred projects are cheap and highly likely while expensive, low odds projects are to be avoided.

Figure 0**??** shows three sets of actions learned by TAR2. The right-hand-side histogram shows the baseline distributions seen in the 2000 simulations. The other histograms show how those ratios change after applying the treatments learned by TAR2; The *worth* of each option is a reflection of the proportion of good and bad projects, compared to the baseline, i.e. ($worth(baseline) = 1$). Note that as *worth* increases, the proportion of preferred projects also increases.

Figure 0**??** shows the three best treatments ($T_1, T_2, T_3$) found using this technique (and Figure 0**??** compared the effects of these treatments to the untreated examples). Note that the values of each attribute are reported using the tags *no*, *lower*, *middle*, or *upper*. In treatment learning, continuous attribute ranges are divided into N-discrete bands based on percentile positions. For N=3, we can name the bands *lower*, *middle*, *upper* for the lower, middle, and upper 33% percentile bands.

In Figure 0**??**, the treatments are advising to lower the cost of:

—*Using requirements:* This could be accomplished by (e.g.) sharing them around the development team in some search-able hypertext format
—*Performing formal reviews at milestones:* This could be accomplished by (e.g.) using ultra-lightweight formal methods such as proposed by Leveson [**?**].
—*Performing good unit testing:* This could be accomplished by (e.g.) hiring better test engineers.

An interesting feature of Figure 0**??** is what is *missing*:

—None of the treatments proposed adjusting the *Chances* of any action. In this study, changing *Cost* will suffice.
—Of the 27 actions listed in in Figure 0**??**, only the four underlined actions appear in the top three treatments. That is, management commitment to undertake 27-4=23 of the actions is less useful than changing on $formalReviewsAtMilestones$, $goodUnitTesting$, $periodicSoftwareReviews$, and $requirementsUsed$
—The value *not* in $T_1$ is a recommendation against *periodicSoftwareReviews* (plus lowering the costs of using requirements and formal reviews at milestones). Note

that if *periodicSoftwareReviews* are conducted, $T_3$ is saying that there is no apparent need to reduce the cost of such reviews.

More generally, in a result consistent with the prior studies, despite the uncertainties introduced by *rany* and the *cost/chances* ranges, TAR2 found a small number of CMM2 process options that have a significant impact on the project.

Note that the conclusions of Figure 0**??** are not general to all software projects. The *Chances* values used in this study came from some local domain knowledge about the likelihood that process change $A$ will effect process change $B$. The *Cost* values were domain-specific as well. In other organizations, with different work practices and staff, those *Chances* and *Cost* values could be very different.

## 6. CASE STUDY 2: REQUIREMENT OPTIMIZATION

This example illustrates the application of treatment learning on requirement optimization via an iterative learning cycle.

Planning for the optimal attainment of requirements is an important early life cycle activity. Such planning is difficult when dealing with competing requirements, limited resources, and the incompleteness of information available at requirements time. The pilot study discussed here is an evaluation of a promising piece of research-quality spacecraft technology. The purpose of the evaluation is to identify the risks that would arise in maturing this technology to flight readiness, and what mitigation could be identified to address those risks in a cost-effective manner.

### 6.1 The Requirement Interaction Model

For the pilot study, NASA experts built a real-world model developed in the Defect Detection and Prevention (DDP) framework [**?**]. The model is a network connecting 32 requirements, 69 risks and 99 mitigations. Risks are quantitatively related to requirements, to indicate how much each risk, should it occur, impacts each requirements. Mitigations are quantitatively related to risks, to indicate how effectively each mitigation, should it be applied, reduces each risk. A set of mitigations achieves benefits, but incurs costs. The main purpose of the model is to facilitate the judicious selection of a set of mitigations, attaining requirements in a cost-effective manner. This kind of requirements analysis seeks to maximize benefits (i.e., our coverage of the requirements) while minimizing the costs of the risk mitigation actions. Optimizing in this manner is complicated by the interactions inside the model - a requirement may be impacted by multiple risks, a risk may impact multiple requirements, an action may mitigate multiple risks, and a risk may be mitigated by multiple actions.

### 6.2 The Iterative Learning Cycle

Our approach is to follow the iterative cycle of simulation, summarization and decision shown in figure 0**??**. The requirements interaction model is used to grow dataset representing the space of options, treatment learner summarizes the data and gives critical decision alternatives (e.g., the control variables and their corresponding settings), the domain experts review the alternatives and make final decisions. This way, experts make more effective use of their skill and knowledge

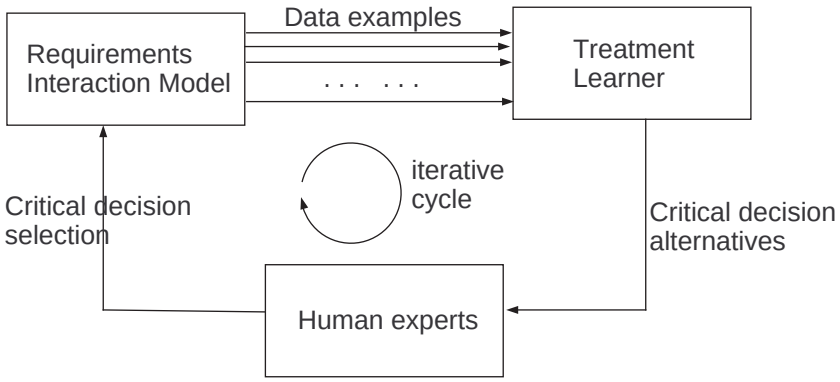


Fig. 8. The iterative cycle of Simulation/Summarization/Decision.

by focusing their attention on the relatively small number of most critical decision alternatives. Repeating this cycle leads the iterative approach to the optimal (or near optimal) decision within the options space.

6.2.1 *Baseline Simulation.* The model was initially executed by selecting risk mitigations at random. This generated 30,000 instances of combinations from the 99 risk mitigations actions. Each instance of the combinations was evaluated by the numerical cost and benefit values automatically computed based on domain data. The study needs to identify the optimal solutions that attain high benefit(approximately 250) while remaining a relative low cost limit(around $600,000). The option space is huge: $2^{99} \approx 10^{30}$ sets of decisions are to be explored. Figure 0**??** shows the initial output of the cost-benefit distribution from the model. The wide spread dots indicate a large variance in the possible cost and benefit ranges.

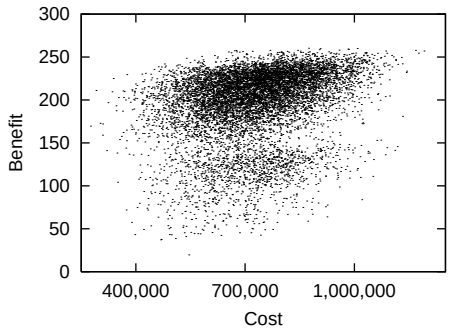


Fig. 9. Initial result from executing the model of pilot domain.

| score | $< \$600K$ | $[\$600K, \$650K)$ | $[\$650K, \$700K)$ | $> \$700K$ |
|---|---|---|---|---|
| high 25% | 16 | 14 | 11 | 7 |
| mid high 25% | 15 | 12 | 8 | 4 |
| mid low 25% | 13 | 9 | 5 | 2 |
| low 25% | 10 | 6 | 3 | 1 |

Table I. Balanced score combination of cost and benefit values

6.2.2 *Combining Cost and Benefit Values.* TAR2 takes dataset containing one single discrete class attribute. We must combine the cost and benefit values into a single score before applying TAR2 on the simulation data. This domain-specific process was proceeded as follows:

—Partitioning cost value into 4 regions: below $600,000 (most desirable region); $600,000 to $649,999; $650,000 to $699,999; at or above $700,000 (least desirable region).

—Partitioning benefit value by subdividing it into quartiles, i.e., putting the lowest 25% of the benefit figures into the lowest benefit range, the next 25% into the next, etc.

—Ranking the 16 possible pairings of cost and benefit according to a balanced scheme which yielded a combined score of "goodness". The scheme is shown in table 0**??**.

6.2.3 *Learning Iterations.* We used TAR2 as a knowledge acquisition tool to summarize the simulation dataset. After ran it on the examples, a set of treatments was discovered and the best was selected by the domain experts. We then imposed the treatments on the model, i.e., some mitigations were to be performed and some were not; others were kept random. Simulating the constrained model again gave us another example set. The whole process was repeated, each run of TAR2 resulted in a new set of constraints, which were then imposed on the model before the next simulation. After five iterations, TAR2 found 30 out of 99 decisions (6 per run. 6 was the maximum size for which it successfully terminated) that significantly effected the cost/benefit distribution. Figure 0**??** shows the model output following the $5^{th}$ iteration. Compared to figure 0**??**, the variation among the cost-benefit figures is relatively small. Since the model represents human experts' estimates, the computed cost-benefit figures should not be misinterpreted to have high precision. At the point where the figures are so tightly clustered, it is appropriate to stop.

The entire series is shown in 0**??**. The first percentile matrix (called **round 0**) summarizes figure 0**??**. The **round 4** corresponds to the dot plotting in figure 0**??**, in which a compact set of points concentrated at the upper end of the benefit range (around 250), and at a cost of approximately $6000. From **round 0** to **round 4**, the variance was reduced and the mean values improved.

## 6.3 Compared to Simulated Annealing

Parallel to treatment learning, a simulated annealing algorithm (SA) was also applied to the same requirement analysis task [**?**]. Simulated Annealing is a commonly used search algorithm for optimization problem. It combines random selection and hill climbing to find global maxima. In particular, it does a random walk, choosing

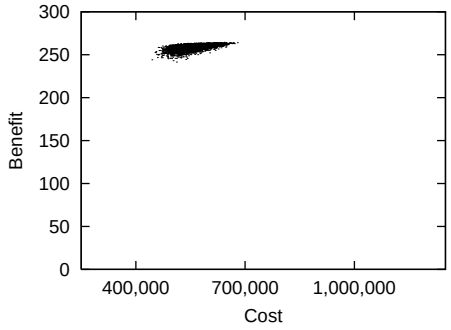


Fig. 10. Result from executing the model of pilot domain when it was constrained by treatments after the $5^{th}$ iteration.

neighbors at random and deciding at random whether to visit that neighbor. The randomness is a function of a "temperature" variable. When $T = \infty$, it chooses neighbors at random; in the limit as $T$ approaches zero, it chooses only neighbors that improve the value. If the temperature is reduced slowly enough, this guarantees to find the global optimal result.

Figure 0**??** compares TAR2 and simulated annealing. At each round X (shown on the x-axis), simulated annealing or TAR2 was used to extract key decisions from a log of runs of the model. A new log is generated, with the inputs constrained to the key decisions found between round zero and round X. Further rounds of learning continue until the observed changes on costs and benefits stabilizes. The comparisons show that:

—As seen in Figure 0**??**, simulated annealing and TAR2 terminate in (nearly) the same cost-benefit zone.

—Simulated annealing did so using only 40% of the data needed by TAR2;

—However, while TAR2 proposed constraints on 33% of the mitigations, each SA solution specifies whether a mitigation should be taken or not for all 99 mitigations. Hence there was no apparent way to ascertain which of them are the most critical decisions. This loses the main advantage of TAR2; i.e. no drastic reduction in the space of options.

## 7. TAR3

TAR3 is a nondeterministic version of the TAR2 data miner [**?**] that exploits narrow funnels to learn very small theories. This section argues that TAR3 is *simple*, *compentent*, *fast*, *scalable* and a *stable* nondeterministic analysis method. Further, TAR3 can be used to detect when *nondeterminism is unsafe*.

### 7.1 An Introduction to TAR3

Given some oracle that scores each state, it is possible to learn some assignments to the funnel variables that improve the average score. Better yet, if the funnels are narrow (few in number) then those experiments won't take long and the required assignments will only be few in number. The key is to find the funnel variables and

**round 0:**

| Benefit \ Cost | 400K | 600K | 800K | 1,000K | Totals |
|---|---|---|---|---|---|
| 250 | | 6 | 15 | 5 | 26 |
| 200 | 1 | 22 | 27 | 4 | 54 |
| 150 | 1 | 6 | 5 | 1 | 13 |
| 100 | | 3 | 3 | | 6 |
| 50 | | 1% | | | 1 |
| Totals | 2 | 38 | 50 | 10 | 100 |

**round 1:**

| Benefit \ Cost | 400K | 600K | 800K | 1,000K | Totals |
|---|---|---|---|---|---|
| 250 | 7 | 45 | 13 | | 65 |
| 200 | 12 | 22 | 1 | | 35 |
| 150 | | | | | |
| 100 | | | | | |
| 50 | | | | | |
| Totals | 19 | 67 | 14 | | 100 |

**round 2:**

| Benefit \ Cost | 400K | 600K | 800K | 1,000K | Totals |
|---|---|---|---|---|---|
| 250 | 9 | 8 | 7 | | 24 |
| 200 | 18 | 58 | | | 76 |
| 150 | | | | | |
| 100 | | | | | |
| 50 | | | | | |
| Totals | 27 | 66 | 7 | | 101 |

**round 3:**

| Benefit \ Cost | 400K | 600K | 800K | 1,000K | Totals |
|---|---|---|---|---|---|
| 250 | 9 | 70 | 11 | | 90 |
| 200 | 3 | 7 | | | 10 |
| 150 | | | | | |
| 100 | | | | | |
| 50 | | | | | |
| Totals | 12 | 77 | 11 | | 100 |

**round 4:**

| Benefit \ Cost | 400K | 600K | 800K | 1,000K | Totals |
|---|---|---|---|---|---|
| 250 | 1 | 81 | 17 | | 99 |
| 200 | | 1 | | | 1 |
| 150 | | | | | |
| 100 | | | | | |
| 50 | | | | | |
| Totals | 1 | 82 | 17 | | 100 |

Fig. 11. Percentile matrices showing four rounds of treatment learning for the pilot study.

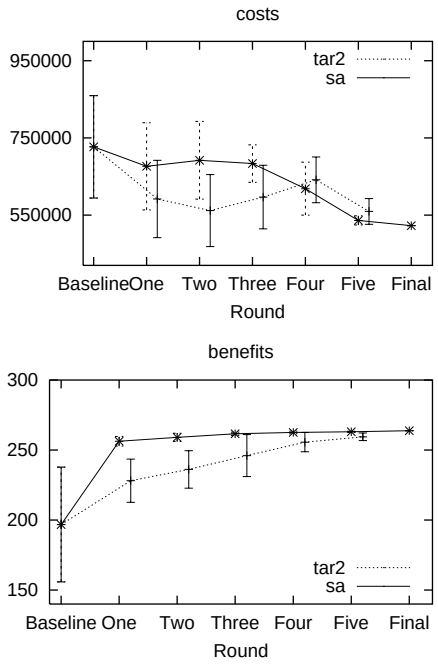


Fig. 12. Comparison of TAR2 and simulated annealing.

| *outlook* | *temp*($^o F$) | *humidity* | *windy?* | *class* | *score* |
|---|---|---|---|---|---|
| *rain* | *85* | *86* | *false* | *none* | *-1* |
| *rain* | *65* | *70* | *true* | *none* | *-1* |
| *overcast* | *71* | *96* | *true* | *none* | *-1* |
| ... | ... | ... | ... | ... | |
| *overcast* | *83* | *88* | *false* | *lots* | *1* |
| *overcast* | *64* | *65* | *true* | *lots* | *1* |
| *overcast* | *72* | *90* | *true* | *lots* | *1* |
| *overcast* | *81* | *75* | *false* | *lots* | *1* |
| ... | ... | ... | ... | ... | |

Fig. 13. Data on some recreational activities where playing *lots* of golf scores higher than playing *none* golf.

this is very simple: just look for a small number of variable assignments that have very different distributions assocaited with the different scores.

TAR3 is a nondeterministic search for combinations of variable assignments that most improve, or *lift*, the expected score. TAR3 inputs a log of data observations $D$ of the behavior of a system such as Figure 0**??**. Each behavior $d \in D$ is scored by some oracle: e.g. see the last column of Figure 0**??**.

TAR3 begins by discretizing all numeric variables into $R$ distinct ranges. Discrete ranges for the score are called the *classes* and each class scores a number value $\$class$; i.e. the mean value of that range. Next, a *baseline* is computed from the weighted sum of the frequency of each class range times the mean score of that class; i.e.

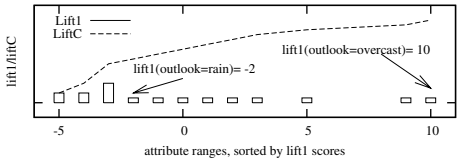


Fig. 14. Lift and cumulative lift.

| | |
|---|---|
| inputs: | R : number of discretized ranges for numeric variables; default=5<br>D : dataset of examples; dataset is of size \|D\|; each example is scored, e.g. Figure 0??; these scores will be divided into R ranges; class(R) is the highest scoring class.<br>ENOUGH : minimum ratio of class(R) required for any treatment<br>VARIABLES : a set of size \|V\| listing the variables used in D;<br>N : maximum number of treatments to return; default=10<br>MAXSIZE : maximum size of each treatment; default=4<br>MAXTRIALS : build this many treatments before pausing to check if anything new has been found; default=100<br>MAXFUTILE : stop after this number of pauses if nothing new found; default=5 |
| outputs: | array of constraints (each constraint is a conjunction of variable assignments) |

```
1  do R quartile discretization of all numeric variables
2  if the score is numeric then do R quartile discretizations of scores fi
3  for {v=r} ∈ range(VARIABLES)
       lift1[{v=r}] ← lift({v=r}) /*see "lift1" in Figure 0??*/
4  compute the cumulative distribution liftC from lift1 /*see "liftC" in Figure 0??*/
5  TREATMENTS ← ∅
6  futile ← 0
7  repeat
8      some ← best ← ∅
9      MAXTRIALS timesRepeat
10     do     rx ← ∅
11            size ← 1 to MAXSIZE, picked at random
12            size timesRepeat rx ← rx ∪ ({v=r} selected at random from liftC)
13            rx.lift=lift(rx) /*calculated using Equation 0?? */
14            if     |rx ∩ class(R)| ≥ (|D ∩ class(R)|∗ENOUGH)
              then   some ← some ∪ rx
       done /* MAXTRIALS repeats */
15     best ← the N top lifters of some
16     if       TREATMENTS contains every member of best
       then     futile ← futile + 1
       else     futile ← 0
                TREATMENTS ←the N top lifters of (TREATMENTS ∪ best)
       fi
17 until (futile ≥ MAXFUTILE)
18 return TREATMENTS sorted on lift
```

Fig. 15. TAR3

$$baseline = \sum_{c \in classes} \$c * |d \in D \wedge d.class = c|$$

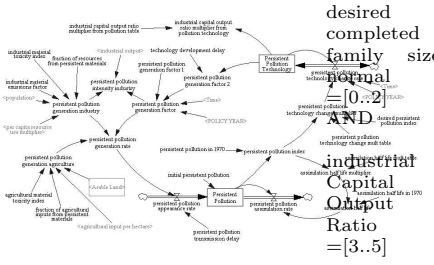

Figure 0??.A: part of the World model.

desired completed family size normal =[0..2]
AND
industrial Capital Output Ratio =[3..5]

Figure 0??.B: TAR3's learnt treatments. Contains two tests.

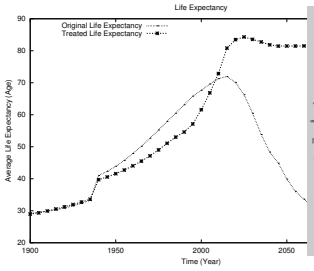


Figure 0??.C: World population 1900 to 2100. The unmarked line is the baseline from the original simulation and the line with dots shows a resimulation after applying the treatments of Figure 0??.B.

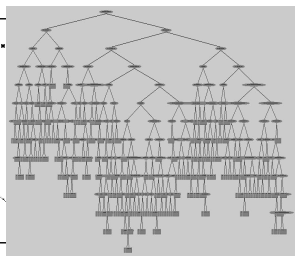

Figure 0??.D: A decision tree learnt by the J4.8 leaner [?]. Contains over 200 tests.

Fig. 16. Output from TAR3, using data generated from the Limits to Growth Model.

After that, the effect of each assignment (denoted $v = r$), is determined by comparing the baseline to the weighted sum of all the classes in examples consistent with the assignment:

$$lift = \sum_{c \in classes} \frac{\$c * |d \in D \wedge d.class = c \wedge (assignment \wedge d) \not\vdash \bot|}{baseline} \quad (2)$$

The $lift$ of a single assignment is denoted $lift1$. If $lift$ is more or less than one then the assignment $v = r$ improves or degrades (respectively) the expected score. For example, in Figure 0??, *outlook=overcast* occurs more frequently when we playing *lots* of golf and *outlook=raining* occurs more frequently when we play *none* golf. Hence, when we compute their *lift1* scores in Figure 0??, we see that *outlook=overcast* has a $lift1$ value larger than 0 and much larger than *outlook=rain*.

The TAR3 algorithm returns *treatments* (i.e. assignments to the narrow funnel variables) by selecting assignments nondeterministically from $liftC$, i.e. the cumulative $lift1$ distribution shown in Figure 0??. The distribution weights the selection process such that assignments with higher lifts tend to be selected more often.

After the discretizations and construction of $liftC$, the algorithm (shown in Figure 0??) attempts MAXTRIALS times to find some good treatments (see line 9 in Figure 0??). Each treatment contains a nondeterministically selected number of assignments (up to a user-supplied MAXSIZE value- see line 11). Each treatment is scored using Equation 0?? (at line 13). In order to avoid over-fitting, a treatment is ignored if it doesn;t select for ENOUGH of the highest scoring class (see line 14).

After MAXTRIALS attempts, the algorithms selects the N treatments with highest lifts. This process is repeated till no new treatments are found after MAXFUTILE number of repeats (see lines 16,17).

## 7.2 TAR3 is Competent

Hu [?] describes numerous studies with TAR3 and datasets from the standard UCI data mining data sets [?] plus some software engineering domains In the usual case,

TAR3 generated treatments that greatly changed the distribution of classes in a data set. See [**?**] for more details.

Other researchers have found TAR3 to be competent at controlling a simulation. Figure 0**??**.A shows part of the Limits to Growth model of global economy [**?**]. This model studies the effects of the world's exponentially growing population and economy. The full model contains 295 variables and over 100 nodes (for space reasons, only a small portion of that model is shown in Figure 0**??**.A). MIT faculty studied this model for several years to find factors that prevented global population over-shoot and collapse. TAR3 found the same factors (shown in Figure 0**??**.B) in a 30 minute run. Most of that time was spent generating the example data from the model- the learning only took seconds. To check TAR3's conclusions, another simulation was conducted where the inputs were constrained according to TAR3's recommendation. The results are shown in Figure 0**??**.C) [**?**]: average life expectancy in the year 2100 increased from 30 to 80 years and appears to be stable from that time onwards.

### 7.3 TAR3 is Simple

Due to the simplicity of the random search, Figure 0**??** is easy to implement. For some years now we have been building data miners. With one exception[5], TAR3 is the simplest data miner we have every built. Before implementation, we had thought we would need to tune various aspects of the algorithm after extensive experimentation. For example, TAR3's discretizaion policy (see line 1 of Figure 0**??**) seemed overly-simplistic and we expected that it would have to be replaced (perhaps with a state-of-the-art entrophy-based supervised discretization policy [**?**]). Howevere, TAR3 proved to be so competent that we were never motivated to revisit its design.

TAR3 is not only simple to build, but it learns very simple theories. Standard data miners are not funnel-aware and can build uncessarily large theories using less-influencial variables. On the other hand, a learner which that assumes narrow funnels can generate far smaller theories than other approaches. For example, when the Limits to Growth simulation data was given to a standard decision tree learner, the result was a decision tree with hundreds of tests (see Figure 0**??**.D). Such large trees are harder to understand than the small treatment found by TAR3 (e.g. Figure 0**??**.B).

The Limits to Growth model is not the only dataset where TAR3's nondetetrministic search yielded a more concise summary of a domain than traditional approaches. In Hu's work with dozens of examples from the standard UCI data mining data sets [**?**], treatments of MAXSIZE$\leq$ 4 were sufficient to effect major changes in the distributions of classes [**?**]. For example, Figure 0**??** and Figure 0**??** compares what a decision tree learner and a treatment learner can learn from the same data set. The data set in those figures comes from Boston housing information. It contains 506 examples of houses with approximately the same number of

[5]Our experience is that Naive Bayes classifiers for discrete attributes are simpler to describe and build than TAR3 [**?**]. Very simple discretization methods allow their extension to numeric attributes [**?**]. However, such classifiers don't offer succinct generalizations of a data set so we prefer TAR3.

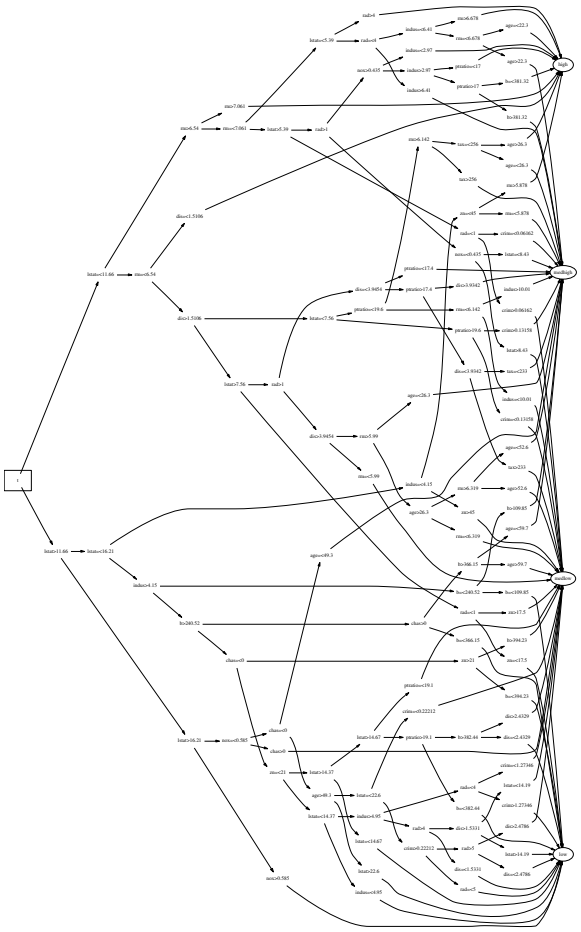

Fig. 17. A decision tree learnt by J4.8 [**?**] decision tree learner from the same data set as Figure 0**??**. Classes (right-hand-side), top-to-bottom, are “high”, “medhigh”, “medlow”, and “low”.

“high”, “medhigh”, “medlow”, and “low” houses (the symbol denotes their median value in $1000’s). A decision tree, learnt via standard methods, needs hundreds of tests in order to distinguish these classes (see Figure 0**??**). However, a little meta-knowledge about class preferences and some nondeterministic search yields a much smaller result. After TAR3 is told that houses score a value, highest to lowest, “high”, “medhi”, “medlow”, “low” (respectively), it learnt the treatment:

$$12.6 \leq ptratio < 16 \wedge 6.7 \leq rooms < 9.78$$

Figure 0**??** shows that this simple output is a powerful predictor for “high” quality houses. This treatment selects a subset of the houses with 97% “high” quality houses (and this treatment is also seen in all sub-samples of a 10-way cross validation).

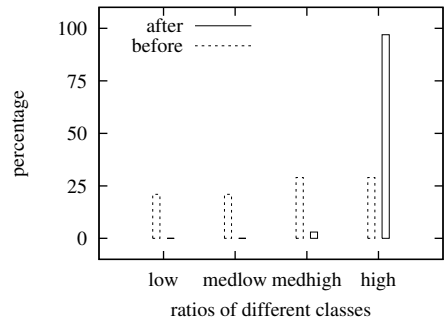


Fig. 18. Ratio of different housing types seen before and after applying a TAR3 treatment. Results from same data as used in Figure 0**??**.

### 7.4 TAR3 is Fast and Scalable

TAR3 has been carefully benchmarked against TAR2, a deterministic version of the sames algorithm [**?**]. TAR2 first prunes all assignments with a $left1$ less than user-specified "promising" value. Next, all the combinations of size MAXSIZE of the remaining assignments are explored [**?**]. In a result consistent with narrow funnels, TAR3's partial nondeterministic search found nearly the same treatments (identical in all cases but one) as TAR2's complete deterministic search.

Other benchmarks show that TAR3 is faster than TAR2, particularly on larger data sets. Further, TAR2's runtimes increase exponentially on MAXSIZE, while TAR3's runtimes increase linearly on number of examples |D| and number of variables |V| [**?**].

### 7.5 TAR3 is Stable

One concern with a nondeterministic search is that it can generate different solutions every time it runs. Paradocially, the reverse if often true: the behaviour of a nondeterministic algorithm can actually be *more stable* than a deterministic algorithm.

It is is easy to see why this might be so. In order to determine a boundary on a deterministic algorithm, it is standard procedure to construct inputs for which the algorithm runs poorly. A nondeterminisitc algorithm algorithms can be viewed as a *probability distribution on a set of deterministics algorithms*. Motwani and Raghavan [**?**] argue that a nondeterministic methods are *less susceptible to problem variations* than deterministic methods. While it may be possible to construct an input that foils one (or a small fraction) of the available deterministic algorithms, it is difficult to find inputs that detect a randomly chosen algorithm.

Certain data mining algorithms are unstable; i.e. minor variations in the order of the input examples can significantly effect the generated theory. Many data mining algorithms depend on this instability. Proponents of bagging and boosting [**?**; **?**] repeatedly sub-sample the available data to generate ensembles of theories. The ensemble is then polled to reach a consensus opinion.

TAR3, in constrast, only returns stable treatments. The tool reports only the treatments found in the majority of a 10-way cross-validation; i.e. are stable across

multiple random sub-samples of a dataset. One reason for this stability might be the Motwani and Raghavan explanation offered above: TAR3's random search is less suceptible to variations in the input data. Another possibility is that TAR3 does not generate very detailed summaries of the input data. Recall from the above that TAR3 generally returns treatments with less than MAXSIZE$\leq 4$ tests while a decision tree learner might return a tree with hundreds of tests (recall Figure 0**??**). Hence, minor details that vary across sub-samples could change the output of a decision tree learner while being ignored by TAR3.

### 7.6 TAR3 and Checking for Unsafe Nondeterminism

Our thesis is that deterministic analysis is more safe than a nondeterministic one if a nondeterministic random sampling yields as much information as a complete deterministic search. In some systems, making choices at random, will survey as mmuch of the system as a complete search. Those systems are the ones with narrow funnels since they contain only a few key choice points (the assignments to the funnel variables).

TAR3 is designed to failure if a small number of outstandingly influencial variables do not reveal themselves to a nondeterministic search. The only way the nondeterministic search of Figure 0**??** will work is if a few variable settings are highly correlated with different states of the system.

In practice, TAR3 has worked very well on all our test domains suggesting that narrow funnels are not uncommon. Nevertheless, if TAR3 *fails* to find influencial variables that would be an indicator that nondeterminism is inappropriate for the current domain,

## 8. CASE STUDY: THE PILOT DOMAIN AGAIN

The above studies dealt with small to medium sized datasets. This section describes an experiment with a very large dataset in which TAR3's treatments, while similar, were not identical.

We have discussed the "pilot" case in requirement optimization domain (see chapter 3). To compare both the performance and experimental results, we ran TAR2 and TAR3 on the same "pilot" domain again. The model used this time is a revised version of the original one, which contains fewer details. The data set obtained after simulation has 58 attributes instead of 99. Each example is also evaluated by a pair of cost and benefit figures. According to the domain experts, we combined cost and benefit into a single attribute using the same balanced scheme but with different dividing thresholds. The combination resulted in 16 classes representing 16 levels of "goodness".

Figure 0**??** shows the initial cost-benefit distribution from the baseline simulation. The data points are widely spread across the possible cost and benefit ranges. Further, most low cost points correspond to low benefit level and high benefit points have high cost values. The desired low-cost high-benefit points are very few: less than 3% of the entire data.

We followed the same incremental learning approach as discussed in the last chapter, namely the following steps:

(1) Ran TAR2 and TAR3 on the baseline (initial simulation) data, and generated

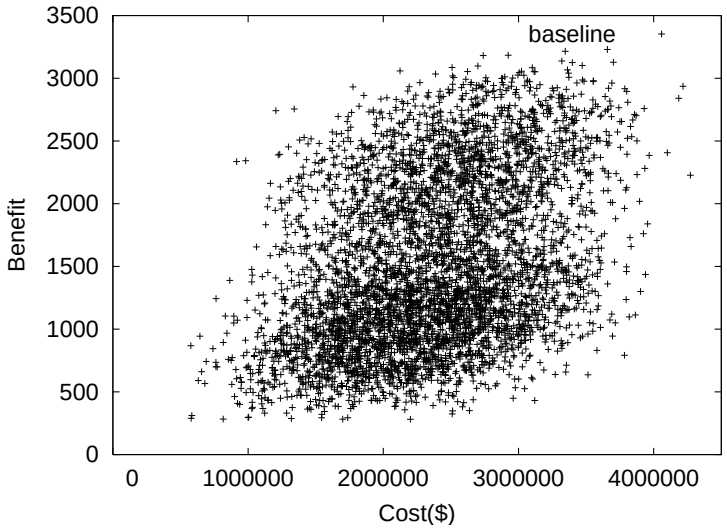


Fig. 19. The cost-benefit distribution of the initial simulation from the pilot domain.

two set of treatments.

(2) The top ranked treatment was chosen from each treatment set. For the purpose of comparison, we didn't ask domain experts to examine the individual treatments, we simply chose the top one.

(3) We then imposed the 2 chosen treatments (1 from TAR2, 1 from TAR3) on the model respectively; simulated it again and got another 2 sets of data examples.

(4) Step 1-3 were repeated until the resulting distribution was so tightly clustered that domain experts agreed to stop.

### 8.1 Comparison of the Cost-Benefit Distribution

Figure 0**??** shows cost-benefit distribution after the $5^{th}$ iteration of TAR2. Compared to figure 0**??**, the variation is relatively small. Most of the data points are grouped at the upper-left corner of the graph, indicating a tight cluster of low-cost high-benefit results. Figure 0**??** is the result from TAR3 experiments following the $4^{th}$ iteration. The two graphs are visually the same, indicating very similar results.

### 8.2 Comparison of the Best 3 Class Distribution

For a closer comparison, table 0**??** records the best 3 class distribution of each round. The best 3 out of total 16 classes correspond to a region of desired zone in which domain experts interested. TAR2 reaches the stopping point after 5 rounds, fixing total 19 attributes; TAR3 reaches the stopping point after 4 rounds, fixing total 20 attributes. At the stopping point, both TAR2 and TAR3 achieved a similar class distribution. Further learning didn't offer significant improvement (i.e., further distribution improvement is less than 5% ).

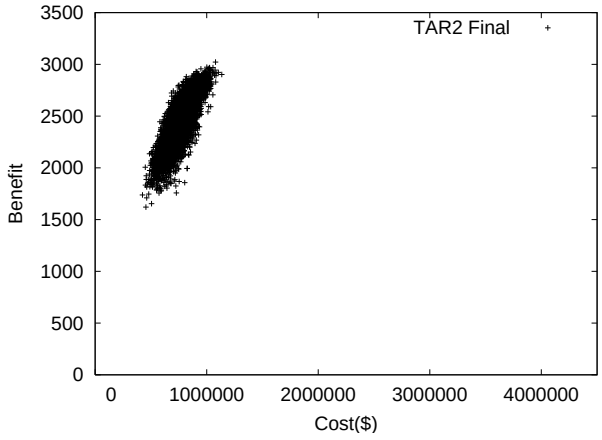


Fig. 20. The cost-benefit distribution from executing the model of pilot domain when it was constrained after the $5^{th}$ iteration of TAR2.

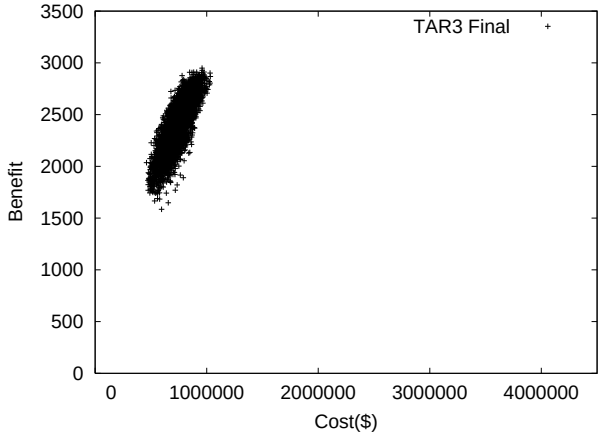


Fig. 21. The cost-benefit distribution from executing the model of pilot domain when it was constrained after the $4^{th}$ iteration of TAR3.

## 8.3 Comparison of Each Round

At the beginning of this experiment, TAR2 and TAR3 started from the same point (i.e. the first baseline data) and came up with different treatments. They later followed their own path toward the final destination. Figure 0**??** compares their performance on each round in terms of the mean and standard deviation of the cost figure. Each round, TAR3 achieved lower mean cost and smaller deviation, allowing it to reach the stopping point one iteration earlier. Figure 0**??** compares the benefit figure. Again, TAR3's deviation is smaller at each round. It is interesting to notice the dip in the TAR2 curve, which indicates a slowing down of the progress. But it eventually catches up in round 4 and round 5.

| TAR2 | baseline | run1 | run2 | run3 | run4 | run5 |
|---|---|---|---|---|---|---|
| size(Rx) | 0 | 4 | 4 | 4 | 4 | 3 |
| Class14 | 3% | 33% | 68% | 22% | 7% | 2% |
| Class15 | 0% | 1% | 7% | 38% | 19% | 5% |
| Class16 | 0% | 0% | 4% | 28% | 74% | 93% |
| Total | **3%** | **34%** | **79%** | **88%** | **100%** | **100%** |
| TAR3 | baseline | run1 | run2 | run3 | run4 | run5 |
| size(T) | 0 | 6 | 6 | 5 | 4 | — |
| Class14 | 3% | 47% | 50% | 11% | 0% | — |
| Class15 | 0% | 2% | 19% | 27% | 7% | — |
| Class16 | 0% | 1% | 13% | 60% | 93% | — |
| Total | **3%** | **50%** | **82%** | **98%** | **100%** | — |

Table II. Comparison of the best 3 class distributions for TAR2 and TAR3 experiments.

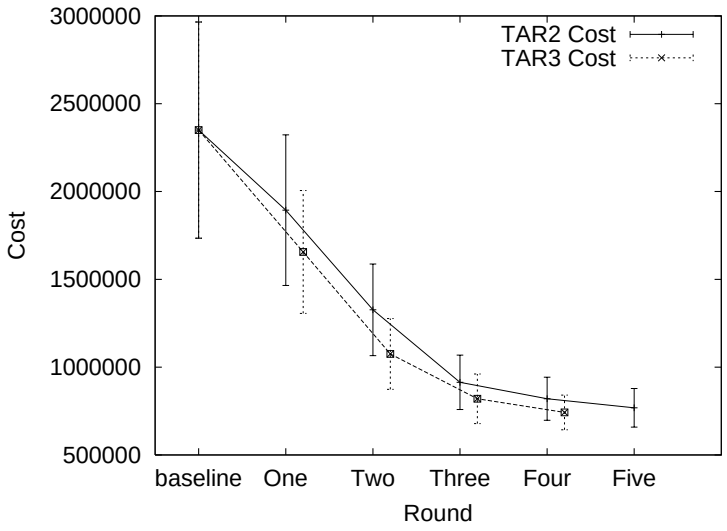


Fig. 22. The mean and standard deviation of cost at each round.

## 8.4 Comparison of the Final Treatments

TAR2 gave a final treatment of size 19 after 5 iterations, TAR3's final treatment is of size 20 after 4 iterations. Although in each run, they generated quite different treatments, the combined final treatments are almost the same. Table 0**??** compares the two final sets attribute by attribute, showing that they have 18 items in common.

## 8.5 Comparison of Runtimes

The data size we used is $20,000$ examples $\times$ 58 attributes at each round. Table 0**??** compares their runtimes. For reference reasons, column 3 and 5 list the size of best treatment found at that round. The average runtimes of TAR3 is only $\frac{1}{5}$ to $\frac{1}{3}$

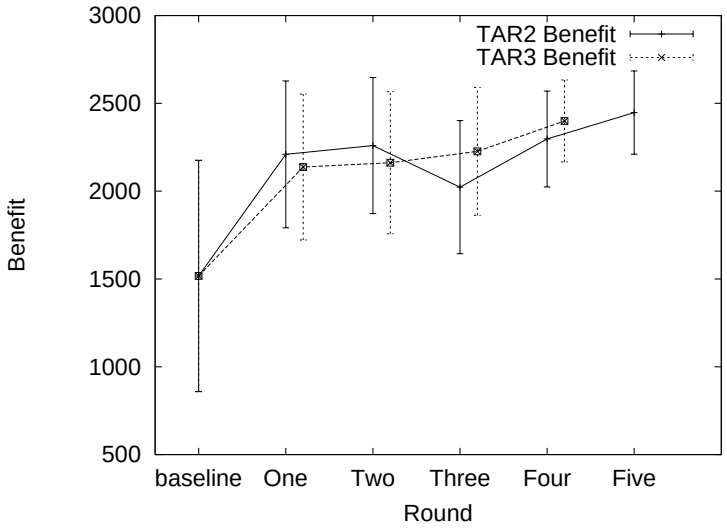


Fig. 23. The mean and standard deviation of benefit at each round.

| No. | Attribute | TAR2 | TAR3 | No. | Attribute | TAR2 | TAR3 |
|---|---|---|---|---|---|---|---|
| 1 | [P63=N] | ✓ | ✓ | 12 | [P1310=Y] | ✓ | ✓ |
| 2 | [P70=N] | ✓ | ✓ | 13 | [P529=N] | ✓ | ✓ |
| 3 | [P72=N] | ✓ | ✓ | 14 | [P544=N] | ✓ | ✓ |
| 4 | [P73=Y] | ✓ | ✓ | 15 | [P551=N] | ✓ | ✓ |
| 5 | [P74=N] | ✓ | ✓ | 16 | [P555=Y] | ✓ | ✓ |
| 6 | [P126=Y] | ✓ | ✓ | 17 | [P575=N] | ✓ | |
| 7 | [P135=N] | ✓ | ✓ | 18 | [P960=N] | | ✓ |
| 8 | [P137=N] | ✓ | ✓ | 19 | [P1047=N] | ✓ | ✓ |
| 9 | [P145=Y] | | ✓ | 20 | [P1260=Y] | ✓ | ✓ |
| 10 | [P154=Y] | ✓ | ✓ | 21 | [P1287=N] | ✓ | ✓ |
| 11 | [P166=N] | ✓ | ✓ | Total | | 19 | 20 |

Table III. Comparison of the final treatments found by TAR2 and TAR3, respectively.

TAR2's runtime. That is, TAR3 ran much faster even with larger treatment size.

| Round | TAR2(sec) | size(T) | TAR3(sec) | size(T) | TAR3/TAR2 |
|---|---|---|---|---|---|
| 1 | 1243 | 4 | 320 | 6 | 25.8% |
| 2 | 1170 | 4 | 348 | 6 | 29.7% |
| 3 | 927 | 4 | 235 | 5 | 25.3% |
| 4 | 650 | 4 | 126 | 4 | 19.4% |
| 5 | 103 | 3 | — | — | — |

Table IV. Comparison of the runtimes of each round.

### 8.6 Summary

From the above case study, we have the following observations:

—In this domain, TAR3 achieved a better class distribution than TAR2 each run, and generated a slightly larger treatment.

—Their own path ended up with a similar yet not identical solution, both in terms of the cost-benefit distribution and the treatment produced.

—TAR2 reached the same final distribution after more runs, but with total less attributes fixed (i.e., the size of the final treatment is smaller in TAR2's case).

—In this domain, TAR3's runtime is much shorter than TAR2, average $\frac{1}{5}$ to $\frac{1}{3}$ TAR2's runtime.

## 9. CONCLUSION

The algorithmic evaluation on TAR2 pointed out situations where its runtimes can grow exponentially. Our solution to this problem is a better learner TAR3. By adopting random sampling together with other strategies, TAR3 has made major improvement in algorithmic efficiency. Experiments have shown that on the datasets where TAR2 is exponential, TAR3 runs in linear time. We have also conducted extensive comparison to survey the stability of TAR3's treatments. It has been seen that TAR3 usually returns identical treatments as TAR2 on small to medium datasets. On high dimensional dataset, TAR3 followed a faster path to goal. The resulting distribution is better, while the final treatment is slightly different.

Specifically, the key idea to treatment learning is the $confidence1$ evaluation of individual attributes. A different search strategy should not change results but only affect efficiency. The sampling method brings in a certain degree of randomness. Still, we have shown that the controlling method we implemented is effective in practice. Given that $confidence1$ distribution represents the probability an item could be picked up in the treatment, there could be other ways to control the random process: For example, some functions could added to the distribution when computing the CDF value.

### 9.1 Discussion

The iterative treatment learning on the pilot study has successfully arrived at a near-optimal attainment of requirements. By identifying only one-third of the mitigations (30 out of 99), we are able to significantly narrow the widely spread cost/benefit distribution.

This case study also demonstrated an incremental use of TAR2: At each iteration, users are presented with list of treatments that have most impact on a system. They select some of theses and the results are added to a growing set of constraints for a model simulator. This approach has two advantages: Firstly, it narrows down the solutions one step at a time, giving a clear statement on which attributes are *most* important; Secondly, the domain experts found this approach user-friendly, since it provided the opportunities for them to inject their knowledge into the process, and allowed them to focus on only a small number of the most critical alternatives.

### 9.2 Other Case Studies

Treatment learning has been applied to spacecraft design to find how to cover more requirements, reduce risk, at the least cost [**?**; **?**]. It has also been applied to software process control using:

—A Chung-Mypolopous soft-goal graph to find better coverage of the non-functional requirements [**?**].
—COCOMO effort and risk models models to find options selecting for lower effort and fewer risks [**?**];
—COCOMO effort, risk, and defect prediction models models to find project options selecting for lower effort and fewer threats and lower defects [**?**];
—Qualitative inference diagrams to find requirements selecting for higher quality [**?**].
—The NASA SILAP model (that selects V&V tasks in order to most lower risks) [**?**];

Treatment learning has also been applied to:

—Finite state machines to find topologies that reduce the CPU cost of applying formal methods [**?**; **?**].
—Models of the global economy so study methods of extending human life expectancy [**?**];
—Maximizing whiskey production [**?**];

In all the case studies explored by TAR2, the same three observations were made:

—Treatment learning can find very small treatments, even for seemingly complex models;
—These treatments can be far smaller that models generated by standard data miners.
—Despite uncertainties or variabilities in the model, TAR2 was able to find effective treatments that selected for preferred model output (but the less uncertainty or model variability, the smaller the variance in TAR2's predicted output for the treated model).

## 10. CONCLUSION

Understanding model configuration options means understanding how input choices affect output scores. That understanding is complex for a certain class of *hard models*; i.e. those with high dimensionality models that are non-linear, non-continuous and built in domains with much noise or other uncertainties, and where managers have limited control over all model inputs.

Hard modeling problems may defeat standard methods. Visualization can't handle very large dimensionality. Analytical methods such as an eigenvector study offer spurious results if the parameters of the variables are uncertain. Other standard automatic methods may be defeated by "cliffs" in non-linear models where the association between inputs and outputs changes abruptly. Data mining methods can handle non-linear models and scale to very large dimensionality. Sadly, data mining techniques like neural nets, genetic algorithms, C4.5 and CART can yield models that are incomprehensible to humans.

TAR2 is a special kind of data miner that produces very succinct output. It assumes that within models there exists a small number of key variables that control the rest. There is much evidence for this assumption. The mathematics of clumps and collars promises that models naturally contain structures that greatly restrict the space of possible model behaviors. TAR2 is a data miner designed to exploit such collars and clumps. It is a minimal contrast set learner that returns a "treatment"; i.e. a minimal, most influential set of deltas between different classes of outcome. The case studies in this paper show that a minimal list of the differences between concepts can be *much smaller* than a detailed description of all aspects of a concept. For models where TAR2 can generate succinct summaries, its algorithm can significantly improved searched-based methods of data mining.

TAR2 addresses the hard modeling problems (discussed in §0**??**) as follows:

—TAR2 offers minimal constraints on the input space and tracks the effects of those constraints, while letting all the other variables vary randomly. Hence, its proposed solutions are not brittle to changes outside the treatments.

—Since it only references a subset of the model inputs, it is a dimensionality reduction tool. In this report, we offered examples where TAR2 reasoned over 100-variable inputs spaces. Elsewhere, we have run it on data sets with over 250 variables. In all cases seen to date, it reduces those spaces to a handful of variables.

—Such small solutions are easier to explain and audit than solutions using all model inputs.

—Further, when managers do not have the budget or authority to control all model input variables, TAR2 can offer them a minimal set which they can use to focus their resources.

—TAR2 has been applied to models with large amounts of noise. For example, in a TAR2-style analysis, we often explore what happens to the solutions when the variance on the model variables increases. Such studies return statements of the form "the solutions offered by this analysis hold for variances up to the following critical threshold values, after which we do not know how to control this domain".

Treatment learning is not indicated for low-dimensionality linear continuous models built in domains that have no noise or other uncertainties, and where managers have full control over all model inputs. Other reasons not to use our tools include when where there is no need to explain or audit models, where reducing model variance is not valuable, and there exists budgets for building and using maximal models.

As to further work, there is no reason to polarize the SE modeling field into "traditional methods" vs "treatment learning". Much could be achieved by combining the two techniques. For example, many of the methods described in §0**??** suffered from the curse of dimensionality. TAR2 could be used as a fast dimensionality reduction tool that could focus a data visualization environment or a sensitivity analysis on the parts of the inputs space that are most crucial. Ideally, that focusing need not wait till the simulation terminates. In *incremental treatment learning*, TAR2 offers feedback during a simulation run into order the guide the simulator

into regions of interest. Before TAR2 can be deployed in that manner, it must be optimized so that it can run fast enough to keep up with the simulator. Currently, we are exploring stochastic methods for that optimization.

June 30, 2007