Optimizing Requirements Decisions With KEYS

Omid Jalali, Tim Menzies
West Virginia University
Morgantown, WV, USA
ojalali@mix.wvu.edu
tim@menzies.us

Martin Feather*
Jet Propulsion Laboratory,
California Institute of Technology,
Pasadena, California, USA
martin.s.feather@jpl.nasa.gov

ABSTRACT

Recent work with NASA's Jet Propulsion Laboratory has allowed for external access to five of JPL's real-world requirements models, anonymized to conceal proprietary information, but retaining their computational nature. Experimentation with these models, reported herein, demonstrates a dramatic speedup in the computations performed on them.

These models have a well defined goal: select mitigations that retire risks which, in turn, increases the number of attainable requirements. Such a non-linear optimization is a well-studied problem. However identification of not only (a) the optimal solution(s) but also (b) the key factors leading to them is less well studied. Our technique, called KEYS, shows a rapid way of simultaneously identifying the solutions and their key factors.

KEYS improves on prior work by several orders of magnitude. Prior experiments with simulated annealing or treatment learning took tens of minutes to hours to terminate. KEYS runs much faster than that; e.g for one model, KEYS ran 13,000 times faster than treatment learning (40 minutes versus 0.18 seconds).

Processing these JPL models is a non-linear optimization problem: the fewest mitigations must be selected while achieving the most requirements. Non-linear optimization is a well studied problem. With this paper, we challenge other members of the PROMISE community to improve on our results with other techniques.

Categories and Subject Descriptors

i.5 [**learning**]: machine learning; d.2.1 [**requirements**]: tools

General Terms

algorithms,experimentation, measurement

Keywords

keys, collars, clumps, DDP

*This was carried out at WVU and CalTech's Jet Propulsion Laboratory under a contract with NASA. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or im- ply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

PROMISE'08, May 12–13, 2008, Leipzig, Germany.

1. INTRODUCTION

Design, said Herbert Simon, is the quintessential human activity [32]. A design optimization method could be used in many domains; software or building construction, car manufacturing, aircraft flight planning, just to name a few. As Simon says "Engineering, medicine, business, architecture and painting are concerned not with the necessary but with the contingent - not with how things are but with how they might be - in short, with design".

Model-based design is now a software engineering technique. Sendall and Kozacaynski argue that increasing productivity and reduced time-to-market for software products can accrue when "using concepts closer to the problem domain ..." via modeling [31]. Hailpern and Tarr observe that model-based design "imposes structure and common vocabularies so that artifacts are useful for their main purpose in their particular stage in the life cycle" [2].

Numerous large organizations now have active model-based SE teams such as Microsoft's Software Factory [16]; Lockheed martin's Model Centric Software Development [33]; the Object Management Group's Common Warehouse Metamodel [5]; and the DDP work at NASA's Jet Propulsion Laboratory [14]. Such models can be queried to find combinations of options that might be otherwise missed. For example, with DDP, the goal is a non-linear optimization that seeks the *least* costly project options that *most* increases the chance of attaining more requirements.

Paradoxically, our prior successes [14, 15, 25] with DDP has caused a problem. Our user community now expects an automatic model-based cost-benefit analysis for larger and larger JPL models containing more variants. Further, as requirements change (as is frequently the case), our users are demanding a complete reanalysis of all past decisions. Extrapolating into the near future, we expect to fall off a computational cliff where our models will be too complex for automatic analysis.

Accordingly, we explore optimizations for model-based design. Prior experiments with simulated annealing [14] or treatment learning [15] terminated in minutes to hours. Our new method, called "KEYS", runs much faster; e.g for one model, KEYS ran 13,000 times faster than treatment learning (40 minutes to 0.18 secs).

The rest of this paper described JPL's DDP modeling systems; our prior work; the new KEYS algorithm; and experiments with KEYS on five DDP models. The intent of this paper is to promote more repeatable experiments in model-based SE. Our models are now available in the PROMISE repository[1]. This paper will be a success if other researchers try alternate methods to find better solutions for the DDP models, or the same solutions in less time.

[1] `promisedata.org/repository/data/models.zip`

2. DDP: SOFTWARE SYSTEMS DESIGN

At JPL, the "Defect Detection and Prevention (DDP)" tool [9,14] is in use to organize interactive knowledge acquisition and decision making sessions with spacecraft experts. The DDP tool provides an ontology for representing these requirements, risks, and mitigations, and for reasoning about them.

DDP might be thought of as akin to the mainstream decision support approach Quality Function Deployment (QFD) [1], but with a quantitative, probabilistic basis inspired by risk assessment techniques. This novel combination places it in a sparsely populated niche in decision making techniques. We believe this is why DDP is useful for studying the requirements needs of a wide variety of technologies, software, hardware and combinations of the two. Quality requirements feature prominently during these studies. DDP takes such requirements into consideration by relating them to "risks" (used very generally to represent all the factors that have the potential to impede the requirements attainment). Then, DDP's support for locating cost-effective risk "mitigation" options can be applied. The net result is an approach that allows stakeholders to explore alternatives among requirements the designs to meet them, and the development approaches to follow, taking into account the costs of those alternatives as they do so.

The process by which DDP is employed is as follows:

- 6 to 20 experts are gathered together for short, intensive knowledge acquisition sessions (typically, 3 to 4 half-day sessions). These sessions *must* be short since it is hard to gather together these experts for more than a very short period of time.
- The DDP tool supports a graphical interface for the rapid entry of the assertions. Such rapid entry is essential, lest using the tool slows up the debate.
- Assertions from the experts are expressed in using an ultra-lightweight decision ontology (e.g. see Figure 1). The ontology *must* be ultra-lightweight since:
  - Only brief assertions can be collected in short knowledge acquisition sessions.
  - If the assertions get more elaborate, then experts may be unable to understand technical arguments from outside their own field of expertise.

The result of these sessions is a network of influences connecting project requirements to risks to possible mitigations; see Figure 2.

The ontology of Figure 1 may appear too weak for useful reasoning. However, in repeated sessions with DDP, it has been seen that the ontology is rich enough to structure and guide debates between NASA experts. For example, DDP has been applied to over a dozen applications to study advanced technologies such as

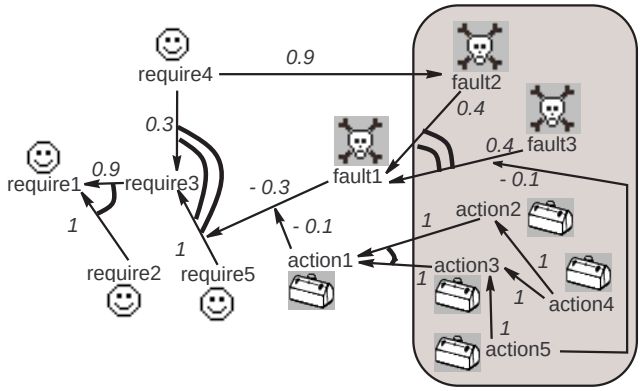
- a computer memory device;
- gyroscope design;
- software code generation;
- a low temperature experiments apparatus;
- an imaging device;
- circuit board like fabrication;
- micro electro-mechanical devices;
- a sun sensor;
- a motor controller;
- photonics; and
- interferometry.

In those studies, DDP sessions have led to cost savings exceeding $1 million in at least two instances, and lesser amounts (exceeding $100,000) in some others. The DDP sessions have also generated numerous design improvements such as savings of power or mass, and shifting of risks from uncertain architectural ones to better understood (and hence more predictable and manageable) design ones. Further, at these meetings, some non-obvious significant

DDP assertions are either:

- *Requirements* (free text) describing the objectives and constraints of the mission and its development process;
- *Weights* (numbers) associated with requirements, reflecting their relative importance;
- *Risks* (free text) are events that damage requirements;
- *Mitigations*: (free text) are actions that reduce risks;
- *Costs*: (numbers) effort associated with mitigations, and repair costs for correcting Risks detected by Mitigations;
- *Mappings*: directed edges between requirements, mitigations, and risks that capture quantitative relationships among them. The key ones are *impacts*, each one of which is a quantitative estimate of the proportion of a requirement that would be lost should a risk occur, and *effects*, each one of which is a quantitative estimate of the proportion by which a risk would be reduced were a mitigation to be employed (the ontology is also able to capture the phenomenon of a mitigation making some risks *worse*).
- *Part-of relations* structure the collections of requirements, risks and mitigations;

Figure 1: DDP's ontology



Faces denote requirements;

Toolboxes denote mitigations;

Skulls denote risks;

Conjunctions are marked with one arc; e.g. *require1* if *require2* and *require3*.

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

Numbers denote weights; e.g. *action5* reduces the contribution of *fault3* to *fault1*, *fault1* reduces the impact of *require5*, and *action1* reduces the negative impact of *fault1*.

Oval denotes structures that are expressible in the latest version of the JPL semantic net editor (under construction).

Figure 2: A semantic net of the type used at JPL

risks have been identified and mitigated. Lastly, DDP can be used to document the information and decision making of these studies. Hence, DDP, although not mandated, remains in use at JPL:

- not only for its original purpose (group decision support);
- but also as a design rationale tool to document decisions.

Note that DDP is not just a software design tool. At JPL, software and hardware are designed together. DDP is best viewed as a *software systems engineering* tool where the interactions between hardware and software can be quickly explored.

3. TREATMENT LEARNING

Discrete models (like DDP) have "cliffs" where the behavior of a system can change dramatically. Such non-continuous models are not suitable for numeric optimization. Hence, in our previous work, we applied data mining to find the least cost set of mitigations that lead to the greatest number of attainable requirements.

The data miner used on DDP was the TAR2 and TAR3 *treatment learners* [7,8,14,15,17,19,23,25]. The premise of treatment learning is that within the space of possible decisions, there exist a small number of key decisions that determine all others. The rest of this section expands on this notion of $keys$, and their implications.

3.1 What are "keys"?

Imagine that a model supports chains of *reasons* that link inputs to desired goals. Some links in the chain clash with others, and some of those clashes are most upstream; i.e. are not dependent on other clashes. In the following chains of reasoning the clashes are $\{e\neg e\}$, $\{g\neg g\}$ & $\{j, \neg j\}$; item the most upstream clashes are $\{e\neg e\}$, & $\{g\neg g\}$,

$$\begin{array}{rl} & a \longrightarrow b \longrightarrow c \longrightarrow d \longrightarrow e \\ input_1 \longrightarrow & f \longrightarrow g \longrightarrow h \longrightarrow i \longrightarrow j \longrightarrow goal \\ input_2 \longrightarrow & k \rightarrow \neg g \longrightarrow l \longrightarrow m \rightarrow \neg j \longrightarrow goal \\ & n \longrightarrow o \longrightarrow p \longrightarrow q \longrightarrow \neg e \end{array}$$

One way to optimizing decision making about this model would be to first decide about the non-dependent clashing links. We call these decisions *the collars* since, as we shall see, they have most impact on the rest of the model.

For example, returning to the above reasoning chains, any of $\{a, b, ..q\}$ is subject to discussion. However, much of this model is irrelevant to the task of $input_i \vdash goal$. For example, the $\{e, \neg e\}$ clash is not exercised in the context of the inputs to the above model. In the context of reaching some $goal$ from $input_i$, the only important discussions are the clashes $\{g, \neg g, j, \neg j\}$. Further, since $\{j, \neg j\}$ are fully dependent on $\{g, \neg g\}$, then the core decision must be about variable g with two disputed values: true and false.

We call g the *collar* since it restricts everything else. The collar may be internal to a model and so may not be directly controllable. A model's $keys$ are the controllable variables that influence the collar. In this example, those keys are $input_i$.

Using the keys to setting the $collars$ reduces the number of reachable states inside a model. Formally, the reachable states reduces to just the cross-product of all the ranges of the collars. We call this the $clumping$ effect; i.e. only a small fraction of the possible states are the reachable states. The effects of clumping can be quite dramatic. Without knowledge of these chains and the collar, the above model has $2^{20} > 1,000,000$ possible consistent states. However, in the context of $input_i \vdash goal$, those 1,000,000 states clumps to just the following two states: $\{input_1, f, g, h, i, j, goal\}$ or $\{input_2, k, \neg g, l, m, \neg j, goal\}$.

3.2 Theoretical Evidence

Theoretically, clumps and collars are the expected average case properties of any model. Druzdel [13] represented a model as the product of distributions of the model's variables, Given enough variance in the individual priors and conditional probabilities of the variables, the frequency of model states will exhibit a log-normal distribution. Such a system would be observed to clump; i.e. a small fraction of states to cover a large portion of the total probability space, with the remaining states having negligible probability.

Menzies & Singh [28] computed the odds of a system selecting solutions to goals using complex, or simpler, sets of preconditions. In their simulations they found that, at a very high probability, a system will naturally select for tiny sets of preconditions (a.k.a. the keys). The Menzies & Singh result means, in theory, that search engines can be greatly simplified since they need only explore for a small number of variables that most effect the rest of the system. The KEYS system discussed in this paper is a perfect example of how simple a search engine can be, yet still be effective. In this regard, the Menzies & Singh paper [28] can be viewed as the *theory* and this paper the *practical application* of the theory.

3.3 Empirical Evidence

There is much empirical evidence for the validity of these theoretical predictions. Keys, collars and clumps have been widely reported (albeit by different names). A sample of those empirical results are offered below (for more, see [28]),

Keys: Numerous researchers have examined *feature subset selection*; i.e. what happens when a data miner deliberately ignores some of the variables in the training data. For example Kohavi and John [22] showed that in numerous datasets, as few as 20% of the variables are key- the remaining 80% of variables can be ignored without degrading a learner's classification accuracy.

Clumps: Druzdzel [13] studied one piece of software where the reached states were a vanishingly small fraction of the set of possible states. In one diagnosis system with 525,312 possible internal states, he found that only one state was ever reached 52% of the time and only 49 states were reached 91% of the time.

Collars: Williams et.al. [34] discuss how to use keys (which they call "back doors") to optimize search. They showed that setting the keys can reduce the solution time of some hard problems from exponential to polytime- provided that the keys can be cheaply located- an issue on which Williams et.al. are curiously silent.

3.4 Finding the keys

A traditional approach to key-based reasoning is to find the funnels using some dependency-directed backtracking tool such as the ATMS [11] or HT4 [24]. Dependency-directed backtracking is very slow, both theoretically and in practice [24]. Further, in the presence of narrow collars, it may be unnecessary. There is no need to *search* for the collar in order to *exploit* it. Any chain of reasoning to goals must pass through the collars (by definition). Hence, all that is required is to find variables that most influence the output score.

One way to find those most influential variables is the "lift" heuristic implemented in the TAR3 treatment learner [19,26] using a table of model output, each column being one input variable, and each row being one run of the model. An extra column adds s_i; i.e. the score assigned to the output of that run by some oracle function. Assuming X rows, and that the scores coming from $|c|$ discrete values with frequency c_i, then this table has a weighted baseline score of $before = \sum_i^c s_i c_i / X$. A particular conjunction of variable values, called the treatment Rx can be used to select for some $Y \leq X$ rows by rejecting all rows inconsistent with Rx. The scores in the selected set contains $|d|$ discrete values with frequency d_i. This

selected subset has a weighted sum of $after = \sum_i^d s_i d_i / Y$

The $lift$ of Rx is $\frac{after}{before}$. The most influential variables are found in the treatments with most impact on improving the score distributions; i.e. those with highest lift. To find those treatments:

1. TAR3 creates one treatment from all variable values;
2. These singleton treatments are sorted by lifts;
3. A $cache$ is created to hold the best treatments;
4. A single rule of size Max is created by selecting randomly from the sorted list of variable values, favoring those with higher lift. Max is selected at random from 1 to some user-specified maximum value.
5. If the new rule has a lift within the top M treatments, add it to $cache$ (and if the cache has now grown beyond size M, delete the worst lifting cached rule).
6. After creating N new rules, if $cache$ has changed, goto 4.
7. Otherwise, return the $cache$ of M best treatments.

Typical values for $\{Max, M, N\}$ are $\{10, 10, 100\}$, respectively.

Formally, TAR3 is a stochastic minimal contrast learner for weighted classes [3, 6]. TAR3 uses a stochastic algorithm since that scales linearly on number of attributes and size of training set [19]. Also, in a result consistent with the prevalence of $keys$, TAR3's fast stochastic search nearly always returns the same treatments as the slower deterministic search of early versions of this algorithm [19].

3.5 Experiments with Treatment Learning

Figure 3 shows one application of TAR3 [15] on a DDP models with 99 possible mitigations; i.e. $2^{99} \approx 10^{30}$ possibilities.

After 10,000 random selections of the mitigations the resulting $costs$ (i.e. sum cost of the mitigations) and $benefits$ (i.e. number of attained requirements) are shown *below* the black line (top left) of Figure 3. All the dots *above* this line represent high benefit, low-cost projects found by iterative applications of treatment learning At each iteration, researchers gave the simulator's output to the treatment learner. TAR3 scored these outputs using the distance of the associated costs-benefits to the "sweet spot" of maximum benefits and minimum costs (the top left corner of Figure 3). Researchers then imposed the top treatment found by TAR3 found onto the simulator for subsequent iterations.

In a result consistent with the DDP models having small $keys$, TAR3 found 30 (out of 99) mitigations that crucially affected cost-benefit. This means TAR3 also found $99 - 30 = 69$ arbitrary decisions that could be made with minimal software impact.

Greenwald [17] benchmarked TAR3 against simulated annealing [21]. TAR3 achieves the same cost-benefit point as simulated annealing [21], but does so using $\approx \frac{1}{10}$-th the evaluations and using $\approx \frac{1}{4}$-th the constraints required by SA. Up until this current paper, Greenwald's results where the high-water mark in learning mitigations for DDP models.

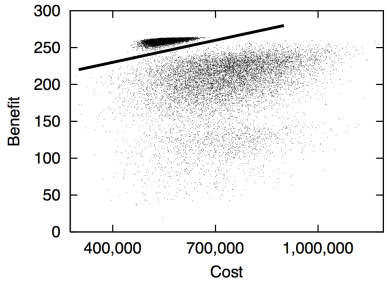


Figure 3: DDP results.

4. KEYS: A FASTER TAR3 FOR DDP

While TAR3 is a useful tool, Figure 3 took 40 minutes to generate. Ideally, our design advisors should run faster than designers could change their models; i.e. orders of magnitude faster than TAR3. Hence, we tried five techniques, discussed below: *greedy search*, *BORE*, *knowledge compilation*, and some *systems tricks*.

4.1 Greedy Search

Step 6 of the TAR3 algorithm (described above) generates hundreds of treatments, then prunes all but the best M. KEYS was an experiment in building one rule per variable, with no post-pruning. KEYS therefore runs much faster than TAR3.

KEYS searches a space of M mitigations in M "eras" (large steps in the search space). Initially, mitigations are free to take any value. At each era, one more mitigation is set to $M_i = X_j$, $X_j \in \{true, false\}$. Each era e generates a sets of $< input, score >$ as follows:

1a: $Selected[1 \ldots (e-1)]$ are settings from previous eras.
1b: $Guessed$ are randomly selected values for other mitigations.
1c: $Input = selected \cup guessed$.
1d: Call DDP to compute $score = ddp(input)$;

After 100 repeats of steps 1a,1b,1c,and 1d:

2: The 100 $score$s are divided into 10% $best$ and 90% $rest$.
3: The mitigation values in the $input$ sets are then scored using BORE (described below).
4: The top ranked mitigation value becomes a setting to one more mitigation and is stored in $selected[e]$.

KEYS then moves to the era $e+1$ and repeats steps 1,2,3,4. KEYS stops when all mitigations have settings.

KEYS slowest step is 1d; i.e. the call to DDP. For a model with 100 mitigations, this call is repeated 100*100=10,000 times. We optimize step 1d using knowledge compilation (discussed below).

4.2 BORE = Best Or Rest

TAR3's lift calculation is a heuristic measure with no theoretical basis. It can favor treatments based on very small portions of the training set. KEYS uses an alternative Bayesian ranking measure, better founded in theory, and one that includes a support measure.

BORE [8] assumes that the output scores are divided into one class for $best$ outcomes and one for the $rest$. In such a two-class systems, TAR3's lift calculation can be replaced with a search for mitigation values that have a high probability of belonging to $best$.

BORE divides numeric scores seen in K runs into $best$ and $rest$, storing the top 10% and the remaining 90% scores (respectively). It then computes the probability that a value is found in $best$ using Bayes theorem. The theorem uses evidence E and a prior probability $P(H)$ for hypothesis $H \in \{best, rest\}$, to calculate a posteriori probability $P(H|E) = P(E|H)P(H)\ /\ P(E)$. Such simple Bayes classifiers are often called "näive" since they assume independence of each variable. Domingos and Pazzani show that the independence assumption is a problem in a vanishingly small percent of cases [12]. This explains the repeated empirical result that seemingly näive Bayes classifiers perform as well as other more sophisticated schemes (e.g. see Table 1 in [12]).

When applying the theorem, *likelihoods* are computed from observed frequencies, then normalized to create probabilities (this normalization cancels out $P(E)$ in Bayes theorem). For example, after $K = 10,000$ runs divided into 1,000 *best* solutions and 9,000 *rest*, the value $mitigation31 = false$ might appear 10 times in the $best$ solutions, but only 5 times in the $rest$. Hence:

$$
\begin{aligned}
E &= (mitigation31 = false)\\
P(best) &= 1000/10000 = 0.1\\
P(rest) &= 9000/10000 = 0.9\\
freq(E|best) &= 10/1000 = 0.01\\
freq(E|rest) &= 5/9000 = 0.00056\\
like(best|E) &= freq(E|best) \cdot P(best) = 0.001\\
like(rest|E) &= freq(E|rest) \cdot P(rest) = 0.000504\\
P(best|E) &= \frac{like(best|E)}{like(best|E) + like(rest|E)} = 0.66 \qquad (1)
\end{aligned}
$$

Previously [8] we have found that Bayes theorem is a poor ranking heuristic since it is distracted by low frequency evidence. For example, note how the probability of E belonging to the best class is moderately high even though its support is very low; i.e. $P(best|E) = 0.66$ but $freq(E|best) = 0.01$.

To avoid such unreliable low frequency evidence, we augment Equation 1 with a support term. Support should $increase$ as the frequency of a value $increases$, i.e. $like(best|E)$ is a valid support measure. Hence, step 3 of our greedy search ranks values via

$$P(best|E) * support(best|E) = \frac{like(best|E)^2}{like(best|E) + like(rest|E)} \qquad (2)$$

4.3 Knowledge Compilation

In knowledge compilation, a theory is compiled off-line into a target language, which is then used on-line to answer a large number of queries, very rapidly [10, 30]. The motivation behind knowledge compilation is to push as much of the computational overhead into the off-line compilation phase, which is amortized over numerous on-line queries; e.g. the thousands of calls to the DDP models made by step 1d of the greedy search.

The SE and AI literature has thus far focused mostly on target compilation languages which are combinations of one or more combinations of conjunctive normal form, state machines, or binary decision diagrams [4, 10]. For our purposes, it was convenient not to use declarative forms. Instead, our knowledge compiler outputs a "C" function.

This knowledge compiler computes and caches a flattened form of the the DDP requirements tree. In standard DDP:

- Requirements form a tree;
- The relative influence of each leaf requirement is computed via a depth-first search from the root down to the leaves.
- This computation is repeated each time the relative influence of a requirement is required.

In our compiled form, the computation is performed once and added as a constant to each reference of the requirement.

For example, here is a trivial DDP model where `mitigation1` costs $10,000 to apply and each requirement is of equal value (100):

$$\overbrace{mitigation1}^{\$10{,}000} \underbrace{\rightarrow}_{0.9} risk1 \rightarrow \left\langle \begin{array}{l} \overbrace{\rightarrow}^{0.1}(requirement1 = 100) \\ \underbrace{\rightarrow}_{0.99}(requirement2 = 100) \end{array} \right.$$

(The other numbers show the impact of mitigations on risks, and risks on requirements).

Our knowledge compiler converts this trivial DDP model into the `model` function of Figure 4. Note that The topology of the network is represented as terms in equations at the bottom of the function. As the topology grows more complex, so do these equations. For example, our biggest model, containing 99 mitigations, generates 1412 lines of `model`.

```
 1 #include "tool.h"
 2
 3 void model(float *cost, float *att, float m[])
 4  {
 5   float costTotal, attTotal;
 6   int oCount = 2;
 7   float oWeight[oCount+1];
 8   float oAttainment[oCount+1];
 9   float oAtRiskProp[oCount+1];
10   int rCount = 1;
11   float rAPL[rCount+1];
12   float rAggrevatedImpact[rCount+1];
13   float rLikelihood[rCount+1];
14   int mCount = 1;
15   int m[mCount+1];
16   float mCost[mCount+1];
18   float roImpact[rCount+1][oCount+1];
19   float mrEffect[mCount+1][rCount+1];
20   mCost[1] = 10000;
21   rAPL[1] =1;
22   rAggrevatedImpact[1]=1;
23   oWeight[1]=100;
24   oWeight[2]=100;
25   roImpact[1][1] = 0.1;
26   roImpact[1][2] = 0.99;
27   mrEffect[1][1] = 0.9;
28   rLikelihood[1] = rAPL[1];
29   /* Mitigation effects on Risk likelihoods */
30   rLikelihood[1] = rLikelihood[1] *
31                    (1 - m[1] * mrEffect[1][1]);
32   /*Risk impacts on Objective attainment proportions*/
33   oAtRiskProp[1] = (rLikelihood[1] *
34        rAggrevatedImpact[1] * roImpact[1][1]);
35   oAtRiskProp[2] = (rLikelihood[1] *
36        rAggrevatedImpact[1] * roImpact[1][2]);
37   /* Objective attainments */
38   oAttainment[1] = oWeight[1] *
39              (1 - minValue(1, oAtRiskProp[1]));
40   oAttainment[2] = oWeight[2] *
41              (1 - minValue(1, oAtRiskProp[2]));
42   attTotal = oAttainment[1] + oAttainment[2];
43   costTotal = m[1] * mCost[1];
44
45   *cost = costTotal;
46   *att = attTotal;
47 }
```

Figure 4: A trivial DDP model

The `model` function is called by step 1d of the greedy search and an array of boolean mitigations `m[]`. It returns the total cost of the selected mitigations (`*cost`) and the number of reachable requirements (`*att`). These two scores are then normalized to a single score s representing the distance to the "sweet spot" of maximum benefits and minimum costs, as follows:

$$s = 1 - \frac{\sqrt{((1 - \overline{cost})^2 + \overline{att}^2}}{\sqrt{2}}$$

Here, $\overline{x}$ is a normalized value $0 \leq \frac{x - min(x)}{max(x) - min(x)} \leq 1$. Hence, our scores ranges $0 \leq s \leq 1$ and *higher* scores are *better*.

The form of Figure 4 is not optimal. Observe that all the computation in lines 5 to 28 is always the same, regardless of what values are passed in with the `m[]` array. Hence, in future versions of this tool, it might be wise to split the `model` function into a `model-setup` function (that only gets called once) and a `model-run` function that uses variables set up by `model-setup`.

This knowledge compiler is not just an algorithms optimization tool. It is also a method that lets JPL retain proprietary information while allowing researchers outside of JPL to access JPL models. The resulting models are anonymized to conceal proprietary infor-

mation, while retaining their computational nature. In our experiments, JPL ran the knowledge compiler and passed to West Virginia University models like those shown in Figure 4. Consequently, JPL could assure its clients that their secrets were safe while, at the same time, allowing researchers outside of JPL to perform experiments like those shown in this paper.

4.4 Systems Tricks

The results of Figure 3 where generated using a Visual Basic version of DDP and a "C" version of TAR3. These ran as separate processes communicating via shell scripts and temporary files. KEYS employs some systems tricks to optimize that rig.

KEYS runs a `make` file to build one "C" program containing the greedy search, BORE, and the `model` generated by the knowledge compiler. This single executable runs in RAM. Consequently:

- The slower Visual Basic code is replaced by faster "C" code;
- The learner and the model can communicate without time consuming disc I/O.

5. RESULTS

KEYS was run on the five JPL requirements models. As shown in Figure 5 models one and three were relatively small and were used to debug KEYS. Models two, four, and five are more interesting. Model 4 was discussed in [27] in detail. The largest, model5 was processed previously by DDP/TAR4 [15].

Model	LOC	Objectives	Risks	Mitigations	Run-Time*
model1.c	43	3	2	2	0.0018
model2.c	260	1	30	31	0.0139
model3.c	58	3	2	3	0.0019
model4.c	1226	50	31	58	0.0906
model5.c	1412	32	70	99	0.1751

*average over 100 runs (in seconds)

Figure 5: Details of Five DDP Models.

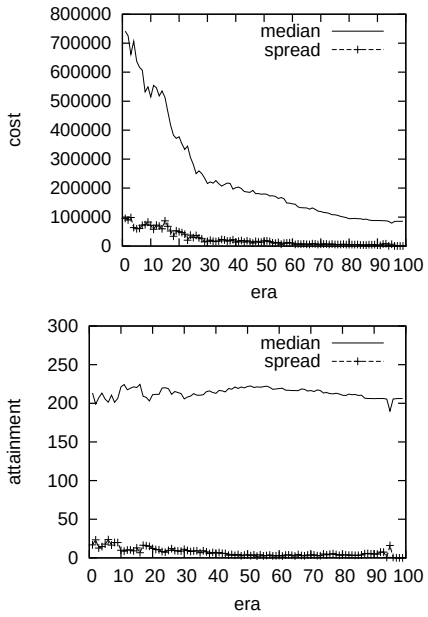


Figure 6: KEYS running on model 5.

The cost-benefits obtained by KEYS were very similar to (or better than) those found by simulated annealing and TAR3. Also, KEYS found those solutions very quickly. KEYS' runtimes (last column of Figure 5) are quite fast: always less than a second, sometimes much less. Better yet, these times are much faster than with TAR3. For example, TAR3 took 40 minutes to process model5 as compared to KEYS' 0.18 seconds ($\frac{0.18}{40*60} \approx 10^4$ times faster).

A typical run of KEYS is shown in Figure 6 (these are results from model5). At $era = 0$, all mitigations are selected at random. At each era after that, one more mitigation is set to true or false. The upper/lower lines in each plot shows median/spread values seen in 100 calls to model5 at each era. Here, "median" is the 50% percentile value and "spread" is a measure of deviation around the median (calculated as 75% percentile value - median). Note that the deviations are quite small, compared to the median. That is, our median estimates are good descriptions of the central tendencies of these models. Also, KEYS reduced the cost while increasing the attainment. Model5 was the exception: attainment remained steady while the cost was greatly reduced.

In Figure 6, the improvements in cost are dramatic up until era=30, after which improvements are much slower. If management wanted a parsimonious set of mitigations, they could hence use the true/false values in the mitigations from eras 0 to 30.

6. DISCUSSION

Perfection is achieved not when there is nothing more to add, but rather when there is nothing more to take away.
– Antoine de Saint-Exupéry

Our results call to mind the "simplicity-first" recommendations of Holte [18]. Sophistication is superfluous if simpler methods perform as well as complex methods. Working with tree-based classification learners, Holte found that extremely small trees were often as accurate as more complex trees. He hence cautioned the machine learning community to benchmark their supposedly more sophisticated algorithms against simpler alternatives.

This study has compared a simple method (KEYS) with a more complex method (TAR3) for solving DDP problems. Not only were the cost-benefit results competitive with (or better than) prior results but the simpler method ran orders of magnitude (10^4) times faster than the complex one.

We attribute most of the success of KEYS to the presence of $keys$ in the DDP models; i.e. a small set of variables that set everything else. These key variables were exploited using two methods:

1. BORE finds promising keys with high support;
2. KEYS' greedy search sets the most promising key, before exploring the remaining options;

We also employed three other methods:

3. Before we used Visual Basic and "C". KEYS just uses "C".
4. KEYS avoids the disk I/O needed by TAR3 talking to DDP.
5. Our knowledge compiler transforms the DDP requirements tree into a set of equations that can be rapidly evaluated.

7. FUTURE WORK

It is unclear which of the above five factors was most influential in speeding up KEYS. In future work, we would like to explore $2^5 = 32$ variants of KEYS that disable some combination of the above five methods. Note that such an exploration is not required

to validate this work- only to isolate the most important features of this implementation which we should retain in all future work.

More importantly, it is important that KEYS runs faster. While the current tools is certainly fast enough for most of the current generation of DDP models, in the very near future, it needs to run much faster. As soon as we give our users more elaborate model-based design tools, they try to build more elaborate designs. Also, as mentioned in the introduction, they want to see a complete analysis of all older designs. Even the 10^4 fold increase in the processing of model5 is not fast enough to handle the requirements models expected in the near future; or the complete set of all past requirement models.

For example, often, families of models are proposed and model-based design methods are asked to assess which subset of, say, 12 alternatives should be used. Allowing 0.2 seconds for each of these 2^{12} models, this would take thirteen minutes to process. Ideally, we seek two second response time (or less) in order to keep up with interactive design discussions. That is, we need at least to run at least 1000 times faster, just to keep up some of the larger design discussions we plan to support in the near future.

Also, if we want to scale KEYS to more models with a more complex ontology, then some modifications may be required. This paper has shown that KEYS has promise for models with binary variables. However, in other work, we have seen disappointing results with models containing variables with much larger ranges.

One approach to speeding up KEYS is to do as much work as possible *before* the model is run. In §4.3, there was some comment along those lines (recall the discussion on splitting the `model` function into `model-setup` and `model-run`).

More generally, there might be a better internal format for the model than "C". The latest generation of stochastic SAT solvers (e.g. MAXWALKSAT [20] or the optimized Markov Logic reasoning within ALCHEMY [29]) use conjunctive normal form. KEYS could be a useful sub-routine of these stochastic tools. For example, stochastic algorithms like MAXWALKSAT and the optimizations inside ALCHEMY all make random choices. Perhaps KEYS could be used to bias random choices towards values that might be part of the $keys$. The effects of such an optimization could be dramatic-recall from the above discussion that Williams et.al. [34] report that setting "back doors" (the keys) can reduce the solution time of hard problems from exponential time to polytime.

8. SUMMARY

In this work we have:

- Described five public-domain real-world requirements models, stored in the PROMISE repository;
- Showed how new methods (KEYS) improve on older ones (TAR3);
- Defined baseline results on the new methods (see Figure 5);
- Proposed a challenge problem for improving these methods (three orders of magnitude faster than Figure 5).
- Offered some suggestions on how to meet that challenge (stochastic search methods like MAXWALKSAT or ALCHEMY, possibly augmented by KEYS to better bias their random selections).

We would welcome collaborations with the PROMISE community on methods to speed up solutions to DDP optimization problem.

Finally, all the software used in this study is free for download and use[2]. We recommend that, where possible, other PROMISE authors offer their results in such a repeatable and improve-able manner.

[2]See `http://unbox.org/wisp/tags/keys/1.0`

9. REFERENCES

[1] Y. Akao. *Quality Function Deployment*. Productivity Press, Cambridge, Massachusetts, 1990.

[2] b. hailpern and p. tarr. model-driven develpment: the good, the bad, and the ugly. *ibm systems journal*, 45(3):451–461, 2006.

[3] S. Bay and M. Pazzani. Detecting change in categorical data: Mining contrast sets. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999. Available from `http://www.ics.uci.edu/~pazzani/Publications/stucco.pdf`.

[4] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *In Proceedings of Tools and Algorithms for the Analysis and Construction of Systems*, page 193207, May 1999.

[5] A. Brown, S. Iyengar, and S. Johnston. A rational approach to model-driven development. *IBM Systems Journal*, 45(3):463–480, 2006.

[6] C. Cai, A. Fu, C. Cheng, and W. Kwong. Mining association rules with weighted items. In *Proceedings of International Database Engineering and Applications Symposium (IDEAS 98)*, August 1998. Available from `http://www.cse.cuhk.edu.hk/~kdd/assoc_rule/paper.pdf`.

[7] E. Chiang and T. Menzies. Simulations for very early lifecycle quality evaluations. *Software Process: Improvement and Practice*, 7(3-4):141–159, 2003. Available from `http://menzies.us/pdf/03spip.pdf`.

[8] R. Clark. Faster treatment learning, Computer Science, Portland State University. Master's thesis, 2005.

[9] S. Cornford, M. Feather, and K. Hicks. DDP a tool for life-cycle risk management. In *IEEE Aerospace Conference, Big Sky, Montana*, pages 441–451, March 2001.

[10] A. Darwiche and P. Marquis. A knowledge compulation map. *Journal of Artifical Intelligence Research*, 17:229–264, 2002. Available from `ww.jair.org/media/989/live-989-2063-jair.pdf`.

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

[12] P. Domingos and M. J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[13] M. Druzdzel. Some properties of joint probability distributions. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 187–194, 1994. Available from `http://www.pitt.edu/AFShome/d/r/druzdzel/public/html/abstracts/uai94.ht%ml`.

[14] M. Feather, S. Cornford, K. Hicks, J. Kiper, and T. Menzies. Application of a broad-spectrum quantitative requirements model to early-lifecycle decision making. *IEEE Software*, 2008. Available from `http://menzies.us/pdf/08ddp.pdf`.

[15] M. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002. Available from `http://menzies.us/pdf/02re02.pdf`.

[16] J. Greenfield and K. Short. *Software factories : assembling*

applications with patterns, models, frameworks, and tools. Wiley Publishing, Indianapolis, IN, 2004.

[17] J. Greenwald. A novel metaheuristic search technique: Iterative treatment learning. Master's thesis, Computer Science, Portland State University, 2007. Available from http://unbox.org/wisp/var/jeremy/Greenwald06Thesis.pdf.

[18] R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63, 1993.

[19] Y. Hu. Treatment learning: Implementation and application. Master's thesis, Department of Electrical Engineering, University of British Columbia, 2003. Masters Thesis.

[20] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications, New York, NY*, pages 573–586, 1997. Available on-line at http://citeseer.ist.psu.edu/168907.html.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[22] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[23] T. Menzies, E. Chiang, M. Feather, Y. Hu, and J. Kiper. Condensing uncertainty via incremental treatment learning. In T. M. Khoshgoftaar, editor, *Software Engineering with Computational Intelligence*. Kluwer, 2003. Available from http://menzies.us/pdf/02itar2.pdf.

[24] T. Menzies and P. Compton. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine*, 10:145–175, 1997. Available from http://menzies.us/pdf/96aim.pdf.

[25] T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from http://menzies.us/pdf/03tar2.pdf.

[26] T. Menzies and Y. Hu. Just enough learning (of association rules): The TAR2 treatment learner. In *Artificial Intelligence Review*, 2007. Available from http://menzies.us/pdf/07tar2.pdf.

[27] T. Menzies, J. Kiper, and M. Feather. Improved software engineering decision support through automatic argument reduction tools. In *SEDECS'2003: the 2nd International Workshop on Software Engineering Decision Support (part of SEKE2003)*, June 2003. Available from http://menzies.us/pdf/03star1.pdf.

[28] T. Menzies and H. Singh. Many maybes mean (mostly) the same thing. In M. Madravio, editor, *Soft Computing in Software Engineering*. Springer-Verlag, 2003. Available from http://menzies.us/pdf/03maybe.pdf.

[29] M. Richardson and P. Domingos. Markov logic networks, February 206.

[30] B. Selman and H. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996. Available from http://citeseer.nj.nec.com/article/selman96knowledge.html.

[31] S. Sendall and W. Kozacaynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, Sept.-Oct. 2003.

[32] H. Simon. *The Science of the Artificial (third edition)*. MIT Press, 1996.

[33] D. Waddington and P. Lardieri. Model-centric software development. *IEEE Computer*, 39(2):28–29, February. 2006.

[34] R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI 2003*, 2003. http://www.cs.cornell.edu/gomes/FILES/backdoors.pdf.