October 11, 2006
WP ref: http://now.unbox.org/all/trunk/doc/06/tellingmore/bayes0.tex
Download this paper at `http://menzies.us/pdf/06anomalies-bayes0.pdf`.

Improving IV&V Techniques Through the Analysis of Project Anomalies: Bayes networks - preliminary report

Tim Menzies

Lane Department of Computer Science and Electrical Engineering, West Virginia University, USA
`tim@menzies.us`

Abstract The original plan of this work was the creation of a process by which conclusions learned from one IV&V project could be applied to another. In particular, we seek methods whereby an agent can say "that's odd"; i.e. detect anomalies and propose repairs in active NASA projects.

Given the current state of business knowledge and IV&V project data recorded at the IV&V facility, the methods proposed in the original plan (semantic web frame-based generalization and specialization over ontologies describing IV&V business practices) are not supportable, Hence, this report describes an alternate direction. Instead of working "top-down" from descriptions of business knowledge (which may never exist), or "bottom-up" from data (that may never be available), this project now focuses on "middle-out" and will try to combine the available data/models into a semantic whole. The currently available data/models are:

- SILAP, from the IV&V planning and scoping team;
- James Dabney's Bayes networks that describe the IV&V business practices of the L3 IV&V contractor;
- The PITS issue tracking data;
- The LINKER database project that intends to join PITS to other data sources;
- Balanced score card strategy maps from NASA Langley.
- and the COCOMO data sets from JPL (note- these data sets are being explored elsewhere; hence, this project will not use its funds to directly explore COCOMO but may lever conclusions from that other project).

This is a three year project that started in June 2006. After five months, and even after the required redirection described above, strong progress has been made. At SAS'06, a preliminary report described what had been learned from the SILAP data. This report presents a preliminary report on our use of the Dabney belief networks. It also offers some background notes on the entire problem.

Subsequent reports will expand these preliminary reports into final conclusions. Those reports will focus on each of the above data sources, one by one, as well as exploring how to combine the above data sources into one anomaly detector.

Acknowledgments: This work would have been impossible without James Dabney's thorough modeling of IV&V processes in belied networks. Also, special thanks are owed to Wesley Deadrick for arranging access to the case study material.

Credits: This research was conducted at West Virginia University under NASA sub-contract project 100005549, task 5e, award 1002193r.

Contents

List of Figures

1 Introduction: From Data to Information

Worms and humans both see data. Worms dig in mud while humans ride rockets to outer space. Why?

Humans can get out of the mud since they use *model-based reasoning* that transforms data into information. Model-based reasoning lets us generalize from the past, and form predictions about the future.

Models let us look before we leap. For every bridge that is built, a hundred more are designed and rejected via mathematical models showing a bridge's statics and dynamics, finite element models, or numerous legislative models that try to maximize for safety/ aesthetics/ etc.

Models not only let us make decisions- they let us audit them as well. Returning to the bridge example, engineers use their models and the calculations they make from those models to justify their decisions.

Models can also be used for training. Newcomers can be shown an organization's business models in order to explain how processes work in that organization

This project is about getting NASA's software development “out of the mud”. Specifically, we seek model-based tools that allow data collected from active NASA development projects to be converted into the information that some project requires urgent management action. Experienced analysts at the NASA IV&V facility already perform this task. Sadly, those decisions are rarely modeled-based. Based on an eight year association with that facility, I assert that IV&V lacks the models that a critical external auditor could use to certify or criticize the decisions made at this facility.

In our target scenario, the IV&V team has access to models connecting what is *observable* within IV&V projects to high-level business *objectives*. The model can be used to raise alerts when newly arrived data shows that the business objectives for a project are under threat.

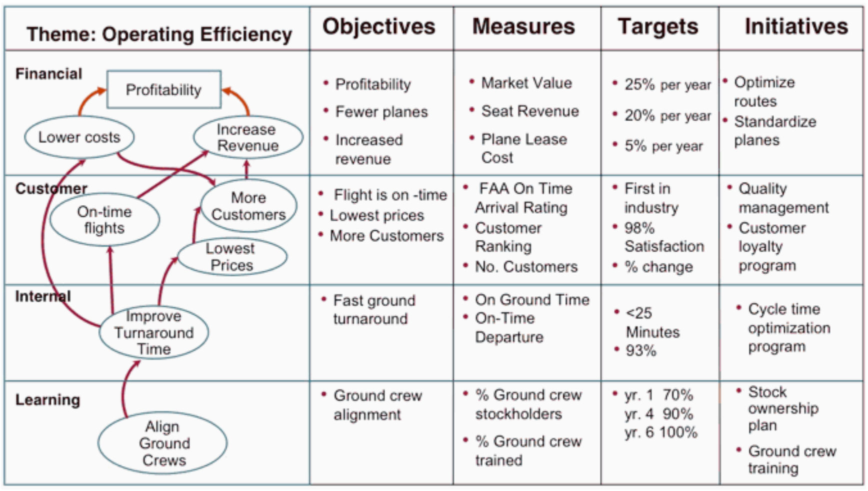
This is a three year project that started in June 2006. This report described half of the work done since then. For the remaining work, see the report on the SILAP data made to SAS'06 and a presentation to the IEEE RE'06 conference on the relative merits of tractability[1].

2 Components

The original vision of this project was “anomaly detection”. However, on reflection, it was clear that anomaly detection is about $\frac{1}{7}$th of what is required.

An anomaly detector is really one of seven components needed to intelligently monitor a device. If this project levers prior and current NASA research projects, then the the following seven components are achievable in the time frame of this project.

[1] At `http://unbox.org/wisp/trunk/silap/doc` see `ebev.pdf` and `re06.pdf`.



Theme: Operating Efficiency	Objectives	Measures	Targets	Initiatives
Financial	• Profitability • Fewer planes • Increased revenue	• Market Value • Seat Revenue • Plane Lease Cost	• 25% per year • 20% per year • 5% per year	• Optimize routes • Standardize planes
Customer	• Flight is on -time • Lowest prices • More Customers	• FAA On Time Arrival Rating • Customer Ranking • No. Customers	• First in industry • 98% Satisfaction • % change	• Quality management • Customer loyalty program
Internal	• Fast ground turnaround	• On Ground Time • On-Time Departure	• <25 Minutes • 93%	• Cycle time optimization program
Learning	• Ground crew alignment	• % Ground crew stockholders • % Ground crew trained	• yr. 1 70% yr. 4 90% yr. 6 100%	• Stock ownership plan • Ground crew training

Fig. 1 A strategy map, left, and a balanced score card, right, modeling business knowledge of a regional airline.

2.1 Data hooks

Without *data hooks*, the anomaly detectors have no raw data to work from.

2.2 Model

Raw data, without some interpretation model, can't be interpreted (by definition). Leake advises that interpretation (which he calls *explanation*) is situation-specific construct that must be tuned to (1) the audience; and (2) the goals of the audience [5]. Hence, prior to the construction of a model that can interpret data, some commitments must be made regarding the who will read the model, and why.

One way to model the audience and their business goals is via *balanced score card* (BST) [4], like the one shown in Figure 1, that contains:

– High level business *objectives*;
– *Measures* that connect raw data to objectives;
– *Targets* values for the variables, as set by defined *initiatives*.

The score card's measures and objectives are then wired up into *strategy map*; e.g. see Figure 1. Score cards and strategy maps are succinct representations that teams of business users can review, audit, improve, and use to train newcomers about local business practices.

Note that without such a *model*, it is hard to know what data should be collected. Further, anomaly detectors can't interpret if an anomaly drastically impacts an important business goal.

At the NASA IV&V facility, as far as I am aware, there is no stated BST-style targets or BST-style initiatives. However, James Dabney's recent ROI study. has generated some knowledge of objectives and measures. Dabney's knowledge is represented graphically in the *belief network* style [7] described in Figure 2.

One advantage of belief networks is that they are intuitively easier for a human to understand direct dependencies and local distributions than complete joint distribution. For example, a sample belief network for requirements defect introduction is shown in Figure 3. In that figure, the objective is $reqQual$ (requirements quality) and the other nodes are the measures (for definitions of each term, see Figure 4). For space reasons, only part of Dabney's networks are shown here- the full system has 161 inputs and 260 nodes in total. With Wesley Deadrick, this project is exploring which of those 260 can be collected via the LINKER data hooks. It is anticipated that only a subset of the nodes will be informed via LINKER. Inference propagation across the networks will be used to fill in, or check, the remaining nodes (see below).

A belief network (also known as a Bayesian network) is a directed acyclic graph of

– nodes representing variables (nodes can represent any kind of variable, be it a measured parameter, a latent variable or an hypothesis);
– arcs representing probabilistic dependency relations among the variables and local probability distributions for each variable given values of its parents.

If there is an arc from node A to another node B, then variable B depends directly on variable A, and A is called a *parent* of B. If there are n variables, then for each variable X_i ($1 \leq i \leq n$), the set of parent variables is denoted by $parents(X_i)$ then the joint distribution of the variables is product of the local distributions

$$Pr(X_1, \ldots, X_n) = \prod_{i=1}^{n} Pr(X_i \mid parents(X_i))$$

If X_i has no parents, its local probability distribution is said to be $unconditional$, otherwise it is $conditional$. If the variable represented by a node is $observed$, then the node is said to be an $evidence$ node.

Fig. 2 About belief networks. From `wikipedia.org`.

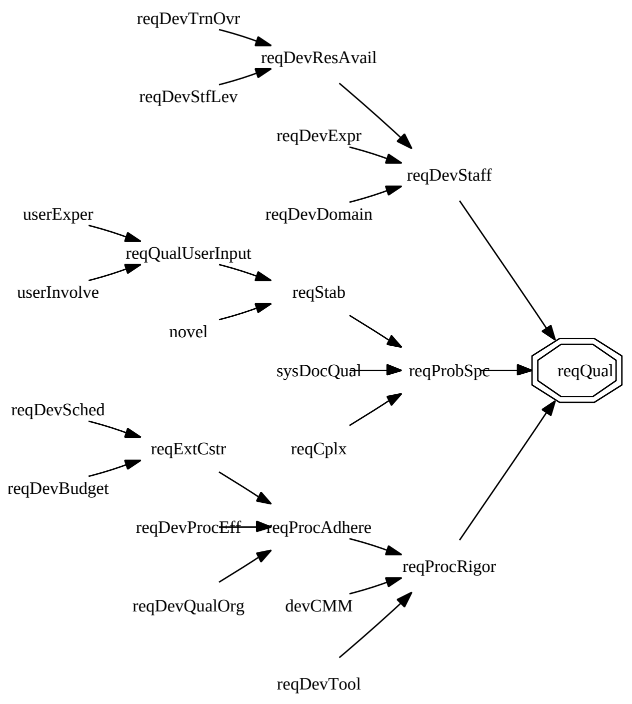


Fig. 3 "Requirements defect introduction": part of the Dabney belief nets. For an explanation of the node names, see Figure 4.

2.3 Calibration

Before users accept the anomaly reports from our models, the models must be *calibrated*, lest the users dismiss our anomaly report with "that model is not relevant to our domain".

2.4 Anomaly detector

Once the above is in place, we can operationalize a *anomaly detector*.

2.5 Fault Localization

The first thing the users will ask is "what is the cause of the anomaly?"; i.e. anomaly detectors need *fault localization* methods.

2.6 Repair

The second thing the users will ask is "how do we fix the anomaly?"; i.e. anomaly detectors need *repair* modules.

2.7 Modeling Tools

It is unlikely that any model we create will appeal to all users. Hence, not only do we need a model, but we also need to give our users *access to modeling tools* to let them create alternate models or modify existing ones. If we do not do so, then we run the risk of reporting anomalies that are irrelevant to certain users.

Currently, the available models are not supported by readily-available modeling tools. Dabney's models require a Matlab license and he has indicated that part of the system requires further funding before he can release it. Also, Dabney has stated that he is unable at this time to support other users working with his tool. Further, the Dabney system was designed to answer certain questions about ROI. Numerous modifications will be required. before it use be used by:

– users and IV&V to refine Dabney's networks or to define new ones;
– anomaly detectors and fault localization algorithms;

Therefore, this project must build its own belief network interpreter. This turns out not to be difficult. The case studies in this paper describes $QNET$ a public-domain belief net interpreter written as part of this project. QNET is distributed under the GNU public license version 2 and is available for download from `http://unbox.org/wisp/trunk/qnet`.

3 QNET

QNET (quality net) lets us represent belief networks like Figure 3 using a Prolog interpreter [2]. The value of QNET is that is it much simpler than alternate belief network implementations (e.g. Dabney's Matlab implementation) and it is fully open source (so it can it be delivered and customized, right now).

As shown in Figure 3, a QNET program starts by loading the $[qnet]$ source code, then defining the $goal$ to be maximized. This is followed the by the definition of the leaf variables (e.g. $devCmm = trig(3, 1, 2)$) and the network that connects the leaves to the goal. Finally, there is a call to $maximize$ to tell QNET to try and maximize the $goal$.

The next section describes the tool used that augments QNET with the calibration, fault localization, etc. This will be followed by a case study. Since this is a preliminary report, the presented case study will be brief (and will just use Figure 3).

3.1 Distributions

QNET connects a set of variables in an acyclic directed graph. The value of each variable is set by its parents. The variables with no parents draw their value from some user supplied *distributions*. Therefore, to understand QNET, two things are needed:

– Rules for setting the user distributions;
– Rules for combining values into parents.

base?	name	notes
y	devCMM	This is an overall assessment of the effectiveness of the development process related to development of requirements. The assessment should include methods for requirements elicitation, coordination, documentation, and validation. It should correlate fairly well with CMM level, but includes an assessment of what is really happening in addition to what is documented.
y	novel	Relative novelty (to the developer or user) of the application/mission or the solution approach. For example, entry GN&C for a new vehicle where all algorithms are adapted from Shuttle would be low novelty (10) (provided the new mission is very similar to the Shuttle mission), completely new algorithms or new application would be high novelty (1).
y	reqCplx	Qualitative estimate of overall system/problem complexity. Related to technical difficulty, required interfaces (developer and system), and unity of users. Not correlated with code complexity metrics. Simple (10) to complex (1)
y	reqDevBudget	How tight is the development budget? Assessment of flexibility in cost growth.
y	reqDevDomain	Average experience of development staff in the specific application domain (e.g., laser guidance system, space telescope, crew rescue, etc). Consider all individual domains within the system (e.g., space telescope will require GN&C, optics, propulsion, system management, telemetry, etc).
y	reqDevExpr	Average experience level of development staff, not specific to the problem domain, but overall experience in software development for the domain type (e.g., real-time embedded flight, financial, ground, manned , etc).
y	reqDevProcEff	How much emphasis does development management place on quality (as opposed to productivity)? For example, does development management participate in reviews, ensure that action items are tracked, and encourage extra analysis of suspected problems? Assessment of effectiveness of process problem reports (like our CPAR to document discrepancies in following the process), process improvement actions, activity of a board to assess effectiveness of process, etc. An assessment of the 'aliveness' of the developer process and attention to making it work to produce better products.
y	reqDevQualOrg	How effective is the embedded quality organization? This measure includes consideration of size, breadth and depth of capability, and level of authority granted to quality organization applied to this project.
n	reqDevResAvail	Resource availability reqDevResAvail = reqDevTrnOvr + reqDevStfLev.
y	reqDevSched	How much margin is in the development schedule? Assessment of flexibility in end date.
n	reqDevStaff	Staff ability reqDevStaff = reqDevExpr + reqDevDomain + reqDevResAvail.
y	reqDevStfLev	Assessment of whether the quantity and distribution of staff is sufficient for the problem space
n	reqExtCstr	External constraint pressure reqExtCstr = reqDevSched + reqDevBudget.
n	reqProcAdhere	Process adherence reqProcAdhere = reqExtCstr + reqDevProcEff + reqDevQualOrg.
n	reqProcRigor	Process rigor reqProcRigor = devCMM + reqProcAdhere + reqDevTool.
n	reqProdSpc	Problem space reqProbSpc = reqStab + sysDocQual + reqCplx.
n	reqStab	Requirements stability reqStab = reqQualUserInput + novel.
n	reqQual	Requirements quality reqQual = reqProbSpc + reqDevStaff + reqProcRigor.
y	reqDevTool	Degree to which the developer uses tools in developing and analyzing requirements. Tools here include requirements management tools (DOORs, for example), traceability tools, simulations, process support, etc.
y	reqDevTrnOvr	Experienced or historical rate of change of staff involved with requirements development
n	reqQualUserInput	Quality of user input. reqQualUserInput = userExper + userInvolve.
y	sysDocQual	Qualitative estimate of the quality (completeness, correctness, and consistency) of system documentation from which software requirements may be derived.
y	userExperR	Experience level of system users with similar (or the same) systems or solution approach. Maturity level of users in understanding technical aspects of system to be implemented and the scenarios in which it will be used.
y	userInvolveR	Degree to which the system users are involved in the requirements definition process and the timeliness of that involvement. Note that a high score here requires involvement or representation of all key system users, not just system operators.

Fig. 4 Some definitions of nodes in the Dabney model. Scored 1 to 10.

```
:- [qnet].

goal = reqQual.

devCMM        = trig(3, 1,2).
novel         = trig(5,1,1).
reqCplx       = trig(3,1,1).
reqDevBudget  = trig(8,2,2).
reqDevDomain  = trig(7,2,2).
reqDevExpr    = trig(6,2,2).
reqDevProcEff = trig(6,2,2).
reqDevQualOrg = trig(2,1,1).
reqDevSched   = trig(2,1,1).
reqDevStfLev  = trig(5,4,4).
reqDevTool    = trig(3,2,2).
reqDevTrnOvr  = trig(5,4,4).
sysDocQual    = trig(2,1,1).
userExperR    = trig(8,1,2).
userInvolveR  = trig(8,1,1).

reqStab          = reqQualUserInput + novel.
reqDevResAvail   = reqDevTrnOvr     + reqDevStfLev.
reqExtCstr       = reqDevSched      + reqDevBudget.
reqQualUserInput = userExper        + userInvolve.
reqDevStaff   = reqDevExpr +reqDevDomain  +reqDevResAvail.
reqProbSpc    = reqStab    +sysDocQual    +reqCplx.
reqProcAdhere= reqExtCstr +reqDevProcEff +reqDevQualOrg.
reqProcRigor = devCMM     +reqProcAdhere +reqDevTool.
reqQual      = reqProbSpc +reqDevStaff   +reqProcRigor.

:- maximize.
```

Fig. 5 QNET: example; representing Figure 3.

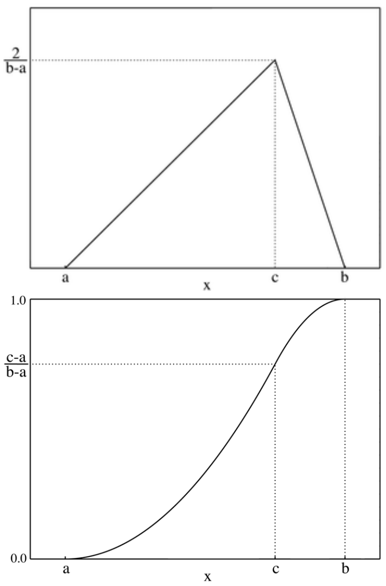


Fig. 6 Triangular distributions: pdf (left) and cdf (right)

3.2 User Distributions

Dabney's belief networks specify the distributions of the leaf nodes as *triangular distributions*. Such distributions are often used in business simulations or project management simulations when there is only limited sample data (e.g. the relationship between variables is known but data is scarce). It is based on a knowledge of the minimum and maximum and an "inspired guess" as to the modal value. Triangular distributions are preferred to Gaussians when there is some imbalance left and right of the mean, or the min/max values are much smaller than offered by a standard deviation.

A sample triangular pdf is shown in Figure 6 (top). Formally, a triangular distribution is a continuous probability distribution with lower limit a, mode c and upper limit b with the ranges $a \in (-\infty, \infty)$, $b > a$, $a \leq c \leq b$ and probability density function (PDF):

$$PDF(x \mid a,b,c) = \begin{cases} 0 & \text{for } x < a \\ \frac{2(x-a)}{(b-a)(c-a)} & \text{for } a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } c < x \leq b \\ 0 & \text{for } x > b \end{cases}$$

These four cases are represented inside QNET as follows:

```
trPDF(A,_,_,X,0) :- X < A,!.
trPDF(A,B,C,X,Y) :- X >= A, X =< C,!,
                    Y is 2*(X - A)/((B-A)*(C-A)).
trPDF(A,B,C,X,Y) :- X >  C, X =< B,!,
                    Y is 2*(B - X)/((B-A)*(B-C)).
trPDF(_,B,_,X,0) :- X > B.
```

Also shown in Figure 6 is the triangular cumulative distribution function (CDF):

$$CDF(x \mid a,b,c) = \begin{cases} 0 & \text{for } x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & \text{for } a \leq x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & \text{for } c < x \leq b \\ 1 & \text{for } x > b \end{cases}$$

These four cases are represented inside QNET as follows:

```
trCDF(A,_,_,X,0) :- X < A,!.
trCDF(A,B,C,X,Y) :- X >= A, X =< C,!,
                    Y is (X - A)^2/((B-A)*(C-A)).
trCDF(A,B,C,X,Y) :- X > C,  X =< B,!,
                    Y is 1 -  (B - X)^2/((B-A)*(B-C)).
trCDF(_,B,_,X,1) :- X > B.
```

The CDF is used when sampling from the distribution. A random number r ($0 \leq r \leq 1$) is generated and the CDF is searched left to right. The returned x-value is the smallest x such that $CDF(x \mid a,b,c) \geq r$.

```
:- arithmetic_function(ranf/0).
ranf(X) :-  X is random(65535)/65536.

tr(A,B,C,_)    :- (C < A; C > B),!,
     print(bad(tr(min = A,max = B,mode = C))),nl,fail.
tr(A,A,_,A)    :- !.
tr(A,B,C,Z)    :- R is ranf, tr1(R,0.01, A,B,C,A,B,Z).
```

Pragmatically, it is useful to search for this x-value using a binary search ($O(log(n))$ time, not $O(n)$). This requires setting some threshold value P such that two numbers are "equal enough" when they are withing P of each other. Here, we use $P = 0.01$.

```
tr1(R,P,A,B,C,Min,Max,Z) :-
    Half is Min + (Max - Min)/2,
    trCDF(A,B,C,Half,Sample),
    myCompare(Op,P,R,Sample),
    tr2(Op,R,P,A,B,C,Min,Half,Max,Half,Z).

tr2(=,_,_,_,_,_,_  ,_   ,_  ,Z,Z).
tr2(>,R,P,A,B,C,_  ,Half,Max,_,Z) :-
   tr1(R,P,A,B,C,Half,Max,Z).
tr2(<,R,P,A,B,C,Min,Half,_  ,_,Z) :-
   tr1(R,P,A,B,C,Min,Half,Z).

myCompare(=, P,X,Y) :- abs(X - Y) =< P,!.
myCompare(Op,_,X,Y) :- compare(Op,X,Y).
```

Dabney he spent much time (and NASA funding) to collect representative distributions from NASA projects; e.g. the $trig$ ranges in Figure 5 come from domain experts at IV&V. Note that Dabney expresses $tr(Min, Max, Mode)$ as

$$trig(Mode, Max - Mode, Mode - Min)$$

Also, our triangular distributions take the range $0..1.0$ while Dabney uses $1..10$. So, in Figure 3, Dabney's $trig(3, 1, 2)$ can be rewritten as $tr(0.2, 0.5, 0.3)$. QNET handles this conversions automatically:

```
goal_expansion(trig(Mode0,Down,Up), tr(Min,Mode,Max)) :-
     Mode is Mode0/10,
     Min  is (Mode - Down)/10,
     Max  is (Mode + Up  )/10.
```

3.3 Combination Rules

The values generated at the leaves (from triangular distributions) are propagated up the network. At each internal node, the results from child nodes are combined then normalized to the range $0 \leq x \leq 1$. Hence, $a = b+c$ is expressed internally in QNET as two lookups for the values of b and c, followed by the normalization step.

```
set(a,Value) --> b = Value0, c = Value1,
       {Value is (Value0 + Value1)/2}.
```

Note that this combination rule is rather simplistic: merely averaging values can loose information about the details of the distributions being combined. In a complete belief network a *conditional probability* table is defined for each node that describes how combinations of values in the parents effect the child. The next version of QNET will allow for the definition of arbitrary combination rules.

QNET automates the conversion of (e.g.) $a = b + c$ into the combination rules shown above. This converter accepts statements of the form

$$y = c_1 * x_1 \pm c_2 * x_2 \pm \ldots$$

(where c_i is the *weight* of x_i) and outputs normalization rules of the form:

```
set(y,Value) --> x1 = Value1, x2 = Value2,...,
     {(Value is Value1 + Value2 + ..)/(c1+c2+...)}.
```

The converter is succinct and easy to modify:

```
xpand(Head=Body0,[got(Head),Rule|Rest]) :-
    xpand1(Body0,Head,Arg,N0,Body,Rest,[]), N is N0,
    expand_term((set(Head,Out) -->
                     Body,{Out is Arg/N}), Rule).

xpand1(A0 + B0, H,A1 + B1, A2+B2,(A,B)) --> !,
    xpand1(A0,H,A1,A2,A), xpand1(B0,H,B1,B2,B).
xpand1(A0 - B0, H,A1 - B1, A2+B2,(A,B)) --> !,
    xpand1(A0,H,A1,A2,A), xpand1(B0,H,B1,B2,B).
xpand1(A * N,   H,P, N   ,        Goal ) --> !,
    [want(A,H)],
    {xpand2(A=X,X*N,P,Goal)}.
xpand1(A / N,   H,P, 1/N ,      Goal   ) --> !,
    [want(A,H)],
    {xpand2(A=X,X/N,P,Goal)}.
xpand1(tr(Min,Max,Mode),_,X,1,{tr(Min,Max,Mode,X)}) --> [],!.
xpand1(trig(Min,Max,Mode),_ ,X,1,{trig(Min,Max,Mode,X)})-->[],!.
xpand1(A    ,   H,P, 1,        Goal  ) -->
    [want(A,H)],
    {xpand2(A=X,X,P,Goal)}.

xpand2( - A = X,    P, -1*P, A=X) :- !.
xpand2(   A = X,    P,    P, A=X).
```

Note one special case: when the rule is a leaf node and references a triangular distribution, the *combination rule* becomes just a look up rule for the leaf value. For example, the first rule in Figure 5 is $devCMM = trig(3, 1, 2)$. After converting Dabney's $trig$ function to a standard tr function, this rule becomes

```
set(devCMM,Value) --> {tr(2,5,3,Value)}.
```

3.4 Caching Assumptions

Consider the network:

```
healthy = trig(7,1,3).
wealthy = trig(8,1,1).
happy   = wealthy + healthy.
serene  = healthy.
goal    = happy + serene.
```

Note that the *healthy* leaf node influences two other nodes *happy* and *serene*. Hence, when exploring the $goal$ of $serene$, we will sample the $healthy$ distribution twice (once from $serene$ and once from $happy$). For a single run, the *same* value should be returned each time; i.e. QNET needs to cache old values before computing new ones.

QNET's cache is a list of pairs $variable = value$ that is passed around between the goals. That is, combination rules like

```
set(a,Value) --> b = Value0, c = Value1,
                {Value is (Value0 + Value1)/2}.
```

are converted internally to rules that pass around a $Cache$ variable:

```
set(a,Value,      Cache0, Out) :-
    =(b,Value0, Cache0, Cache1),
    =(c,Value1, Cache1, Cache),
    {Value is (Value0 + Value1)/2}.
```

Here, $b = Value0$ became $= (b, Value0, Cache0, Cache1)$. QNET's "=" procedure controls the cache:

```
=(X,Y,Cache,Cache) :-          # case 1
    member(X=Z,Cache),!,Y=Z.
=(X,Y,Cache0,[X=Y|Cache]) :-   # case 2
    clause(set(X,_,_,_),_),!,
    set(X,Z0,Cache0,Cache),
    ones(Z0,Z),Y=Z.
=(X,Y,Cache,[X=Y|Cache]) :-    # case 3
    Y is random(1000)/999.

ones(N0,N) :- within(0,1,N0,N).
```

Its three clauses handle the three cases of interest:

1. If X is in the cache, just use the old value.
2. If there exists a rule to compute X, then call it (remembering to pass the Cache to it).
3. Otherwise, generate a random number.

In case one, the cache does not change. In cases two and three, a new value is added to the head of the cache.

3.5 Summary

The above code fragments show the semantic core of QNET. The full system isn't much longer- just 243 lines of Prolog.

The ability to build small, easily customizable, systems is pointless unless those systems can do something useful (see the case study, later in this paper).

4 Learning from the Output

QNET generates case studies. If called N times, it will generate N case studies. This can lead to a data analysis problem since an analyst might be buried under an overwhelming amount of information.

To simplify the analysis, we therefore combine QNET with a data miner. Outputs from QNET are classified into *best* or *rest* using the BORE alogrithm. The TAR3 data miner is then used to find the least number of changes to the inputs that most effect the output. TAR3 was previously developed using NASA funding.

The rest of this section describes BORE and TAR3.

4.1 Multi-Dimensional Optimization using "BORE"

BORE inputs instances scored on multiple utilities and classifies each of them "best" or "rest". BORE maps the instances outputs into a hypercube which has one dimension for each utility. In the case studies that follow, the output will scored on only one dimension (requirements quality) but, in the future, we will score based on quality, minimal deviation from known behaviors, etc etc.

These utilities are normalized to "zero" for "worst", and "one" for "best". The corner of the hypercube at 1,1,... is the *apex* of the cube and represents the desired goal for the system. All the examples are scored by their normalized Euclidean distance to the apex.

For each run i of the simulator, the n outputs are normalized to the range $0..1$ as follows:

$$N_i = \frac{X_i - min(X)}{max(X) - min(X)}$$

The Euclidean distance of $\{N_1, N_2, ...\}$ to the ideal position of $\{N_1 = 1, N_2 = 2, ...\}$ is then computed and normalized to the range $0..1$ as follows:

$$W_i = 1 - \frac{\sqrt{N_1^2 + N_2^2 + ...}}{\sqrt{n}}$$

W_i has the following properties:

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

Fig. 7 TAR3: Playing golf.

- $0 \leq W_i \leq 1$.
- The *higher* W_i, the *better* the run.
- That is, improving W_i can only be achieved by increasing *all* of the utilities.

To determine the "best" and "rest" values, all the W_i scores were sorted. The top BEST% are then classified as "best" and the remainder as "rest".

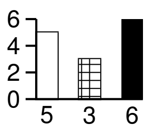
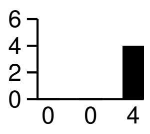
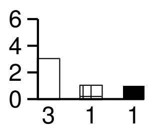
4.2 Treatment Learning with TAR3

Once the above models run, and BORE classifies the output into *best* and *rest*, a data miner is used to find input settings that select for the better outputs. This study uses the TAR3 data miner since this learning method return the *smallest* theories that *most* effect the output. In terms of software process changes, such minimal theories are useful since they require the fewest management actions to improve a project.

TAR3 inputs a set of training examples E. Each example maps a set of attribute ranges to some class symbol; i.e. $\{R_i, R_j, ... \rightarrow C\}$ The class symbols $C_1, C_2..$ are stamped with some utility score that ranks the classes; i.e. $\{U_1 < U_2 < .. < U_C\}$. With E, these classes occur at frequencies $F_1\%, F_2\%, ..., F_C\%$. A "treatment" T of size X is a conjunction of attribute ranges $\{R_1 \wedge R_2 ... \wedge R_X\}$. Some subset of $e \subseteq E$ are consistent with the treatment. In that subset, the classes occur at frequencies $f_1\%, f_2\%, ...f_C\%$. TAR3 seeks the seek smallest T which most changes the weighted sum of the utilities times frequencies of the classes. Formally, this is called the $lift$ of a treatment:

$$lift = \frac{\sum_C U_C f_C}{\sum_C U_C F_C}$$

For example, consider the log of golf playing behavior seen in Figure 7. In that log, we only play *lots* of golf in $\frac{6}{5+3+6} = 43\%$ of cases. To improve our game, we might search for conditions that increases our golfing frequency. Two such conditions are shown in the `WHERE` test of the select statements in Figure 8. In the case of `outlook= overcast`,

input:	`SELECT class FROM golf`	`SELECT class FROM golf WHERE outlook = 'overcast'`	`SELECT class FROM golf WHERE humidity >= 90`
output:	`none none none none none some some some lots lots lots lots lots lots`	`lots lots lots lots`	`none none none some lots`
distributions:			

legend: ☐ none ▦ some ■ lots

Fig. 8 TAR3: Class distributions selected by different conditions in Figure 7.

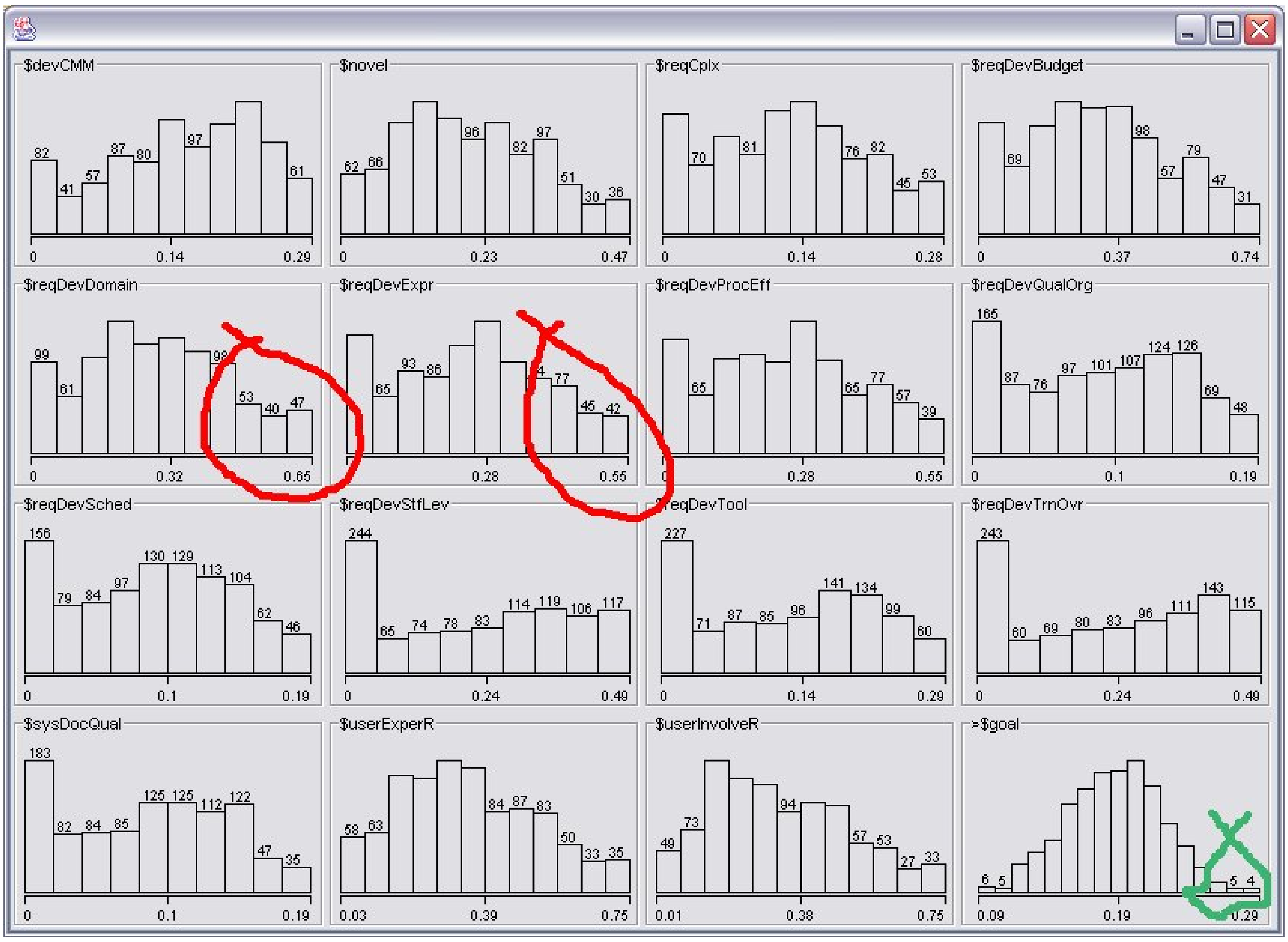


Fig. 9 Case study 1. The green output is best explained by the *combination* of all the red inputs.

we play *lots* of golf all the time. In the case of `humidity` $\leq$ `90`, we only play *lots* of golf in 20% of cases. So one way to play lots of golf would be to select a vacation location where it was always overcast. While on holidays, one thing to watch for is the humidity: if it rises over 90%, then our frequent golf games are threatened.

The tests in the `WHERE` clause of the select statements in Figure 8 is a treatment. Classes in treatment learning get a score U_C and the learner uses this to assess the class frequencies resulting from *applying a treatment* (i.e. using them in a `WHERE` clause). In normal operation, a treatment learner does *controller learning* that finds a treatment which selects for better classes and reject worse classes By reversing the scoring function, treatment learning can also select for the worse classes and reject the better classes. This mode is called *monitor learning* since it finds the thing we should most watch for. In the golf example, *outlook = 'overcast'* was the controller and $humidity \geq 90$ was the monitor.

Formally, treatment learning is a weighted-class minimal contrast-set association rule learner. The treatments are associations that occur with preferred classes. These treatments serve to contrast undesirable situations with desirable situation where more of the outcomes are favorable. Treatment learning is different to other contrast set learners like STUCCO [1] since those other learners don't focus on minimal theories.

Conceptually, a treatment learner explores all possible subsets of the attribute ranges looking for good treatments. Such a search is impractical in practice so the art of treatment learning is quickly pruning unpromising attribute ranges. This study uses the TAR3 treatment learner [3] that uses stochastic search to find its treatments.

5 Case Study

Figure 9 shows the kind of analysis supported by this system. After 1000 samples of the leaf nodes, the inputs and the $goal$ (i.e. $reqQual$, a.k.a. requirements quality) scores were passed to TAR3 to find the *fewest* changes to the current situation that *most improved* $reqQual$. Here, *most improved* was defined to be the top 15% of the *reqQual* scores (shown in Figure 9 as a green circle).

It turns out that of the 16 factors that influence $goal$ ($reqQual$), management attention should be most focused on just two factors. The machine learner found that driving $reqDevDomain$ and $reqDevExpr$ to the top $\frac{1}{4}$ of their range (shown in red in Figure 9) is sufficient to select for the most improvement in *reqQual*. Both these factors relate to the development experience in the particular problem domain (e.g. space telescopes) and in the domain type (e.g. flight systems).

This kind of conclusion can start a detailed discussion of management options. One question that might arise from the above is as follows: "ok, you are telling us that our people have to be experienced- but just how good do our people have to be?". To answer this question, we run the system again but this time we tell the learner that we aren't divided the inputs into four ranges, but only two. That is, we are asking the learner "relax, don't be so strict".

In this second case, it turns out that such relaxation is counter-indicated. If allow a project to be staffed using any of the top-half of our people (measured in terms of development and domain experience). then we will be forced into taking other management actions. As before we have to increase $reqDevDomain$ and $reqDevExpr$, but now we *also* have to improve the software maturity levels and demand a higher level of documentation from which the system will be derived (see the $devCMM$ and $sysDocQual$ results, circled in red, in Figure 10).

6 Discussion

The start of this article defined seven criteria for a useful anomaly detector. This section reviews those criteria.

6.1 Data Hooks

This paper has been silent about the data hooks issue. In the future, this project will hook into the LINKER database, currently under development. The details of those hooks will be discussed in a report due December 31, 2006.

6.2 Model

This paper has modeled IV&V knowledge using belief networks. Subsequent reports will offer cases studies on the full Dabney belief networks.

6.3 Calibration

BORE+TAR3 can perform prediction (as in the above case study) *and* offer calibrate the model. Recall from the above that BORE is a multidimensional optimization method. If we score each node by their distance to known values *as well as* quality, then a single run of TAR3 could optimize for a certain value while moving the distributions of other values towards their known positions.

6.4 Anomaly Detection

Figure 9 and Figure 10 show the distributions in input and output nodes of Figure 3. If we collect distributions on *all* the nodes in our network, then we can build an anomaly detector as follows.

The probability that some value falls at a certain point in the distribution is the area under the curve at that value. By multiplying together all those probabilities, we can compute the likelihood of a particular instance. Note that all the terms in this product are less than one. Hence, if two or three of these values move to a low probability region, the total product will drop by an order or two of magnitude. A monitor could watch for that drop and call a fault localization tool to explain that drop.

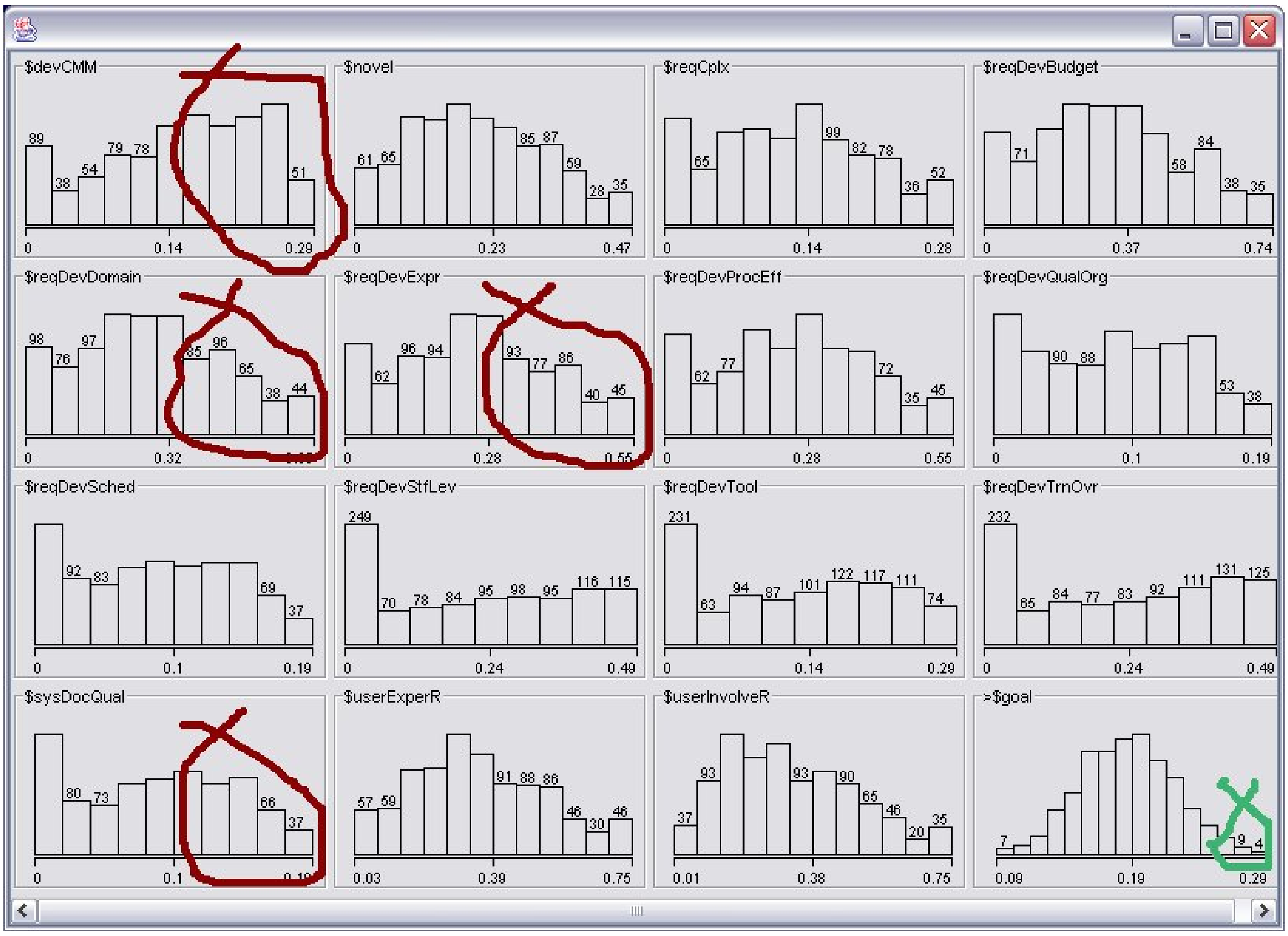


Fig. 10 Case study 2. The green output is best explained by the *combination* of all the red inputs.

Figure 11 shows this kind of anomaly detector operating on data from an F-15 flight simulator. The large drops on the right-hand-side all occurred when five different errors where injected into five different simulations. In all cases, the moment that the anomaly appeared was very clear (note the two orders of magnitude drop in monitored variable). Elsewhere we have confirmed that this anomaly detector works in 27 other data sets [6]. In later reports from this project, we will check if we can detect anomalies in software engineering data sets.

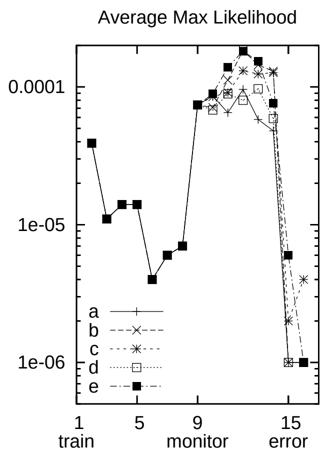


Fig. 11 Anomaly detection.

6.5 Fault Localization

Without any modification, QNET+BORE+TAR3 can be used for fault localization. All that is required is a small change to the scoring function applied to each run of QNET: instead of trying to maximize $goal = reqQual$ (as done in case studies 1&2), TAR3 could be used to find explanations of (say) $goal = reqQual$ being low.

To demonstrate that, imagine that the observed quality is "medium low" to "very low". In Figure 10, , such a quality range could be modeled as $goal \approx 0.15$. The $goal$ outputs from QNET are then replaced by the distance of $goal$ from 0.15 (i.e. $abs(0.15 - goal)$). TAR3 is then asked to find treatments that *minimize* this distance. Another way to say that is as follows:

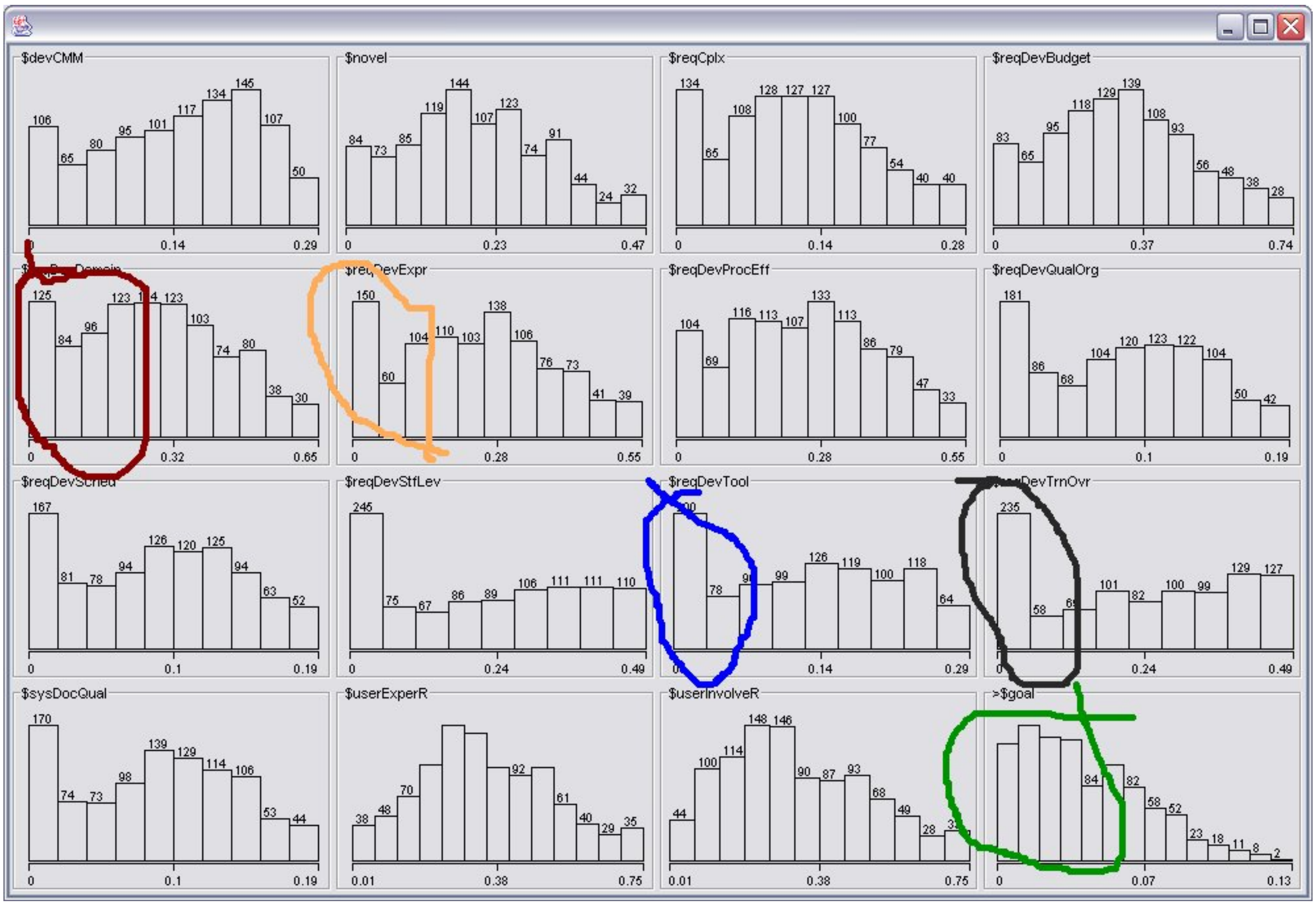


Fig. 12 Case study3 : fault localization. The green output is best explained by *any of* the inputs shown in red, orange, blue, black.
.

- Let low quality be the fault;
- Let fault localization be the factors that lead to that fault.

Figure 12 shows that TAR3 found four alternate theories that explain low requirements quality. Two of them (low $reqDevDomain$ and low $reqDevExpr$) are the reverse of our previous conclusions regarding what most *improves* requirements quality. However there are two new conclusions: either of low $reqDevTool$ or $reqDevTrnOvr$ could explain a low $goal = reqQual$.

Why did case studies one and two return one theory and this fault localization study return four? The answer can be found in the topology of Figure 3. Observe how this graph is a single parent network that connects many inputs (on the left) to a single goal (on the right). Case studies one and two returned one theory since it reasoned from many inputs to a single goal. However, this fault localization study reasoned in the opposite direction. The bad news is that there many factors that could result in low requirements quality (and this is why this case study resulted in multiple theories). To narrow down the true fault, test engineers would now have to collect more information from the projects in order to narrow down the four alternate hypotheses of Figure 12.

On the other hand, the good news is that of the 15 possible causes, there are only four singleton theories that seem most likely to cause low quality. Hence, our test engineers only have four issues to explore, and not the $2^{15} = 32,768$ possible faults supported by Figure 3 (i.e. all combinations of all possible inputs).

6.6 Repair

In this framework, repair is trivial to implement: just ask the treatment learner how to change some undesired state towards a desired state.

6.7 Modeling Tools

As shown in this report, QNET supports an interesting range of modeling tools. Prior work on IV&V process improvement (Dabney, Raffo, Eickelmann) was very heavy on initial model construction and very light on subsequent analysis. The above case studies show that QNET+BORE+TAR3 can be quickly reconfigured to support numerous different kinds of analysis.

7 Figure Work

7.1 Better conditional probabilities

The current QNET system make simplistic assumptions about how children effect parents in the network. This must be changed- in QNET v2.0 it will be possible to specify the combination rules on a node-by-node basis.

7.2 Scale up

The current results, while promising, are only for for a small part of the belief networks specified by Dabney. Further, the use only one of the three case studies documented by Dabney. Subsequent work needs to focus on larger models and more case studies.

7.3 Add data hooks

While the current results are promising, they are not informed by real data from live projects. We look forward to the link to LINKER.

8 Conclusion

The ability to do model-based reasoning seperates humans from worms. Models lets look before we leap; to poke around our ideas and plan our actions *before* squandering scarce resources on a bad idea.

Once a community agrees on a shared vision of their domain, then a variety of powerful tools can be employed for calibration, anomaly detcion, fault localization, and repair. The results here assume that IV&V can be modeled using belief nets. It is hoped that the current results will prompt IV&V civil servants to spend some time in modeling their processes in the notation of QNET or belief networks.

References

1. 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`.
2. I. Bratko. *Prolog Programming for Artificial Intelligence. (third edition)*. Addison-Wesley, 2001.
3. Y. Hu. Treatment learning, 2002. Masters thesis, Unviersity of British Columbia, Department of Electrical and Computer Engineering. In preperation.
4. R. Kaplan and D. Norton. *The Balanced Scorecard: Translating Strategy into Action*. Harvard Business School Press. Boston, 1996.
5. D. Leake. Goal-based explanation evaluation. *Cognitive Science*, 15:509–545, 1991.
6. T. Menzies, D. Allen, and A. Orrego. Bayesian anomaly detection (bad v1.0). In *Proceedings of the Machine Learning Algorithms for Surveillance and Event Detection Workshop, ICML'06*, 2006. Available from `http://menzies.us/pdf/06bad.pdf`.
7. J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–88, 1986.