



# RECAPTURING THE HIGH GROUND USE OF APL IN DECISION TREE MODELLING

Dick Bowman  
Central Electricity Generating Board  
85 Park Street  
London SE1  
England

## INTRODUCTION

In the heyday of APL timesharing APL was frequently used as the implementation vehicle for many types of computational modelling; there were a few bona-fide financial modelling packages but for true flexibility people developed their models using programming languages and APL was in there as good a claim as any (if not better).

But the world moved on; users became able to purchase microcomputers fairly readily, software packages like the spreadsheets gave them some ability to develop their own models (debatedly this was an ability they'd always had - it was just that the spreadsheet industry marketed the users own abilities). And, what with improvements in the mainframe modelling packages we have an area of the computing market which has drifted away from APL - certainly the APL share hasn't grown at anything like the pace of the whole pie.

What this paper does is to examine a recent application of APL to a specific type of modelling, laying emphasis on use of neglected facilities of APL, the useability of some of the more recent APL extensions, evolution of a system from open prototype to final delivered system, the inherent richness of a mainframe timesharing environment and user interfaces which involve graphics.

The theme of the paper is that if we can exploit the power of APL to the full we can develop APL systems which are more attractive to the ultimate users than some of the alternatives, we can define and fill niches which lie outside the scope of the simple package solution, and do all of this without incurring the sort of development overhead consequent in employing more 'conventional' computing solutions. The paper expands on this theme by outlining the evolutionary path taken in

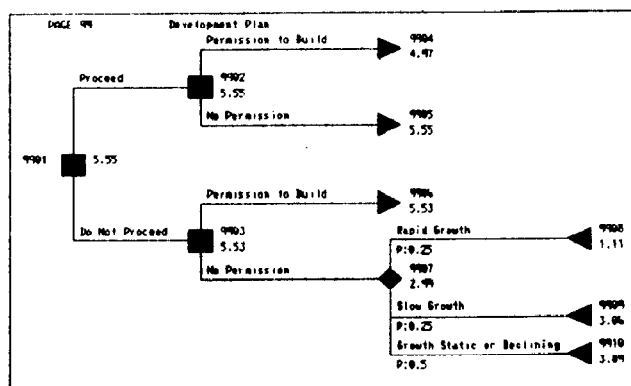
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 0-89791-226-8/87/0005/0427 75¢

one specific development.

## DECISION TREES

Decision Trees are an established analytical tool for examining the decision-making process; they impose a quantification and a structure onto the process and if variation of certain parameters is possible we are able to assess the robustness of choices at particular stages in the process. Here's part of a tree...



Note that this is an extract from the structure of a real-world example, but that all values and text are totally fictitious.

The tree contains four types of node:

- End nodes - Each end node represents an ultimate scenario (the complete set of end nodes may or may not encompass all possible final states of the decision universe). We have a number of attributes by which we can assess this node, and by a combined weighting of these attribute scores we can generate a 'utility' for the end node.
- Probability nodes - We may not have control of our environment, being able only to assign probabilities to specific branches. The utility at a probability node is the weighted sum of those of its branches.
- Decision nodes - These are the places where we can make a choice, the rationale is that we will always elect to choose the branch with highest utility, so this is the utility of a decision node.
- Join nodes - Joining nodes are a convenience mechanism enabling trees to be split into

sub-trees for display purposes only; they are not a distinct type for computation purposes.

Typically, but not compulsorily, the top node of a tree is a decision node; but there can be additional decision nodes further down the tree.

#### ALTERNATIVE APPROACHES

When we first began looking at this technique we found that established practitioners were using equipment like the IBM PC, there were one or two commercial packages we could adopt and that researchers in the area were working with languages like FORTRAN and BASIC. Our criteria were a little wider than "purchase a package, plug it in, off you go". We were examining the whole field of 'decision analysis' (which really isn't the same thing as 'decision support' - even before the marketers turned it into an advertising slogan) - we wanted to establish it as a decision-guidance technique which we had faith in and which we could promulgate widely around the organisation. So, what we were after comprised:

- Insight - In many ways this was the most important attribute, we wanted to know exactly how the tools we were using worked.
- Accessibility - Although 'buy a PC' is a popular saying, we have a large in-house timesharing network and many more of our potential users already have IBM 3270-type terminals than PCs.
- Uniformity - We wanted to build a set of decision analysis tools (this is just one of them) which shared commonality of user interface and which could intercommunicate.

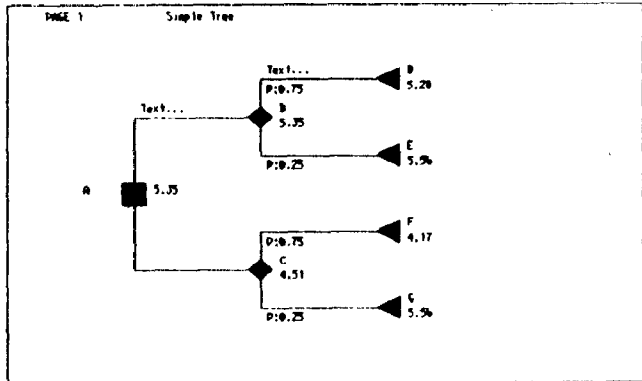
The environment chosen for initial development was our inhouse APL2 service which runs under TSO; APL2 had been running for approximately six months and was replacing a longstanding VSAPL/VSPC service.

#### DIRECT DEFINITION

Direct definition had been in use within the organisation for some time as a method of teaching APL - the fundamental setup (and implementation algorithm) is as per Iverson [1], with some extensions along the lines suggested by Metzger [2]. Some domesticity has been draped around the basic framework with experience showing that:

- Direct definition of functions is non-restrictive in the sense that it can be used to implement non-trivial systems
- It imposes a 'goodness of structure' implicitly (because you have to work harder to be untidy than to be tidy)
- There are no real performance problems
- New users take to it quite well and produce good results using it, but don't really believe that they're using 'proper APL'
- Experienced APLers can be quite violently opposed, even to its use as a teaching tool.

As it happens, decision trees can be defined quite simply in direct definition terms:



Let's work on the tree above, with endnodes scored on three attributes. We need a weighting of the attributes:

WEIGHTS: .25 .25 .5

Each end node has a score:

SCORE\_D: 2 1 8

Leading to an end node utility:

NODE\_D: WEIGHTS+.XSCORE\_D

At probability nodes we have a set of branch probabilities:

PROB\_B: .75 .25

Which in turn allows us to define a utility for the probability nodes:

NODE\_B: PROB\_B+.XNODE\_D NODE\_E

And at the decision nodes the utility is the greatest of the incoming:

NODE\_A: r/NODE\_B NODE\_C

For exploration/display purposes it's useful to be able to attach some descriptive text to each node - easy enough at end nodes:

SHOW\_D: 'Text...' NODE\_D

Higher in the tree we need to display the structure of the tree a little more clearly:

SHOW\_B: 'Text...' NODE\_B (VERT SHOW\_D SHOW\_E)

VERT is the relatively trivial:

VERT: {1,0}w

And that, essentially, is that; given that we approach the task methodically we can build up any decision tree we like, plug in numbers as we want and look at what happens. Adding in the DISPLAY function from supplied workspace 1 DISPLAY [3] gives the user an explicit view of how the tree is made up.

Reviewing where we've got to so far:

- a) We've done about two hours work
- b) Given time and patience we can build trees of arbitrary complexity and size (c.f. limits on branching and depth found in commercial PC solutions)
- c) Everything seems to work without the fanfares about 'folding back the tree' which I never understood when I saw them
- d) Our user interface isn't exactly jovial and unforgiving - but it can be learnt by rote
- e) The system makes no claims to being computationally optimised, but we're looking at problems where wrong decisions cost millions - the cost of modelling exercises is trivial in comparison
- f) We have no real analytical tools to help in exploring decision robustness.

As an aside, we quickly found that 'high good, low bad' scoring systems didn't necessarily correspond to the users' natural perception of the world - a much more acceptable way to score is on a 'low, high, optimum' scheme where compliance with the optimum is an ideal. We need 'high-low' utility, and hence a mapping algorithm between the two schemas. This proved a situation where we gained insight which would not have been achievable if we had merely purchased an off-the-shelf solution; we had extensive discussion of the various score-mapping alternatives, all of which have shortcomings which we might feel unhappy about at one time or another. The final outcome is that we offer a selection - our favourite (and default) being the somewhat obvious:

```
Q10+1          A Throughout...
ISCORE:0f111-(1a[3]-w)+(a[3]-a[1])f[a[2]-a[3]
```

```
      0 10 8 ISCORE 7
.875
```

#### USE OF APL2 NOTATION

Strange as it may sound, things were thrust in the direction of our end user at this stage; they had had no previous encounter with APL but were proficient users of other computing tools and capable of programming in languages like FORTRAN. Essentially what they were told is 'this is how you define a tree' in terms of the functions needed for each type of node, that it was sensible to start at the bottom and work up, and if they did it that way then they could test what they'd done as they went along.

Building block functions like VERT and ISCORE were introduced as 'black boxes'; the users were encouraged to take a 'look and try' approach to using these and the functions they defined themselves to gain confidence in what they were building (as a convention, functions intended to be visible/available to the user have been given uppercase names - system internals use underlined/lowercase naming).

There was a similar approach taken to / and +.x as defined functions; they could have been hidden but were left out in the open.

The unfussiness of vector notation was exploited,

allowing the user to construct objects with quite deep nesting without becoming involved in knowing what they were up to in APL terms.

This was quite frankly an experiment, not just to see what I could get away with - but to what extent we could rely on being able to use raw APL when things became more developed. It worked out pretty well, the type of user we have for this sort of system wants it to work, they appreciated seeing under the bonnet, it demonstrated that APL could do useful work for them on a short timescale, and generated the expected request to do something about the tiresomeness of entering the model.

#### EVOLUTION INTO A CLOSED SYSTEM

At this juncture there are two objectives:

- a) Make the system so that it won't fall over if you blow on it
- b) Extend the capabilities so that insightful analysis of the decision process is facilitated.

Taking the first first, there's no shortage of documentation for APL systems with the historic question-answer dialogue protocol either printed [4] or in one of the many ASK workspaces. We seem to be rather less well-stocked with specific (i.e. copyable/stealable) examples of full-screen utilities; maybe it's because there's so much diversity, or maybe it's a gap that the APL community ought to fill.

Anyway, my instinct was to ignore the dire warnings of [3] and plunge in with the well-tried routines of 2 FSC126; they'd served well in the past and if perhaps a little unexciting, using them had proven a deal more satisfactory in conversion from VSAPL to APL2 than the 'real quick optimised' workspaces that shared variables directly with AP124 (some people only learn by their mistakes). Their use is enhanced by a hermetic layer of user-proofing and an example of how this is eased by recent APL extensions is the SGETNUM function:

```
DCR 'SGETNUM'
Z←N SGETNUM FLDS
Z←DEC, ' ',SGET FLDS      A Hope to get lucky
+((1L+Z)↑ERR             A A Real thumbs job
+(N#p,↑0Z)↑ERR           A Wrong number
Z←↑0Z
→0
ERR: (↑FLDS) SCUR 1 1
SALE
Z←SREA
→1
```

Remember that APL2 has no OFI/OVI functions - one is sceptical about the performance implications of the 'try it and hope' approach which seems to be encouraged by this hardware vendor, but it is at least pragmatic.

Implementing tree input via the consequent fullscreen interface makes sure that:

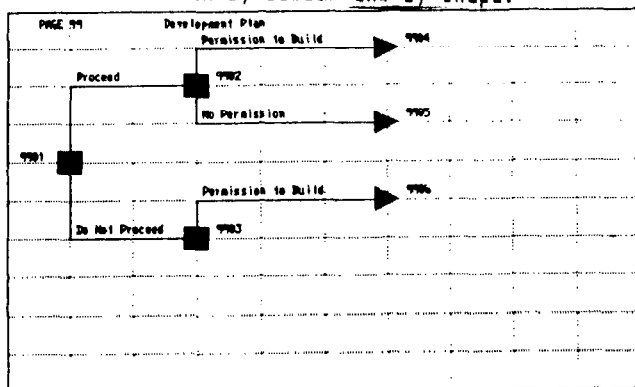


displays helping highlight areas of interest but the user still had a slight gulf in so far as they had a tree which externally took the form of a diagram and a computer system which spoke entirely in terms of form-filling and tables.

This stage was reached coincidentally with the advent of the IBM3179B terminal and 3852 hardcopy unit, devices which delivered graphics capability (including useable hardcopy) to a users desk at a realistic price - we could contemplate replacing ageing 327B terminals with the 3179B on a quite large scale.

The end result is a system which combines pure graphics panels, pure alphanumeric panels and combined graphics/alphanumeric panels - a development which has enhanced the useability of the system and its ability to highlight areas of interest within models by a similar degree to the earlier growth from the hair-shirt prototype. What is provided in detail is:

Input - Following an initial panel soliciting general parameters for the tree (scoring attributes, score profiles, weights, etc.) the user is guided through tree construction by a split graphics/input panel; parameters for each node are gathered through the input fields at the base of the panel and as data for each node is collected the user defines its display position on the tree diagram which is being simultaneously constructed in the upper part of the panel. The system knows what connects where and so all input operations can be seen in the context of what's gone before. Node types are identified both by colour and by shape.



Definition of Node  
Node Type  
Descriptive Title  
Names of Branches  
Probabilities

9907  
P  
No Permission  
9908 9909 9910  
.25 .25 .5

There are a few problems which need to be taken into consideration:

Mice - May be nice, but you need to provide a little assistance if the user is aiming to build up a tidy diagram; what is done in this case is to have a 'snap grid' with actual mouse positions being interpreted as nearest snap point. Users like whizzing mice around, and they get a tidy picture into the bargain.

The screen's smaller than the problem - Isn't it always. There were two alternatives, either a flexible window onto a larger

universe or modular trees; pragmatically the second option was chosen. The 'join node' is a simple one to define - it merely refers to the more explicit definition of the subsidiary page.

Diagram editing facilities are provided, allowing the user to put a tree onto the system quickly and later edit its appearance to a tidier or more acceptable form.

Output - Colour-highlighted tables tell the story, particularly if they're examined closely; but graphs to it more quickly. Alternative approaches were:

- Link into the GDDM's ICU
- Link to another graphics package
- An *ai-fresco* integral development.

Option (a) would have the advantage of uniformity of style with much other graphics produced within the organisation, as would (b); so we initially chose to go down path (c) and put together some graphics specifically for this system.

A bizarre idea, but the context was that received wisdom from others involved in the decision analysis area was that unadorned ordinality was psychologically more acceptable than apparent precision. By removing any possibility of tagging the graphs with specific value we could emphasise some of the important messages of what decision tree modelling is trying to say about the problem at hand:

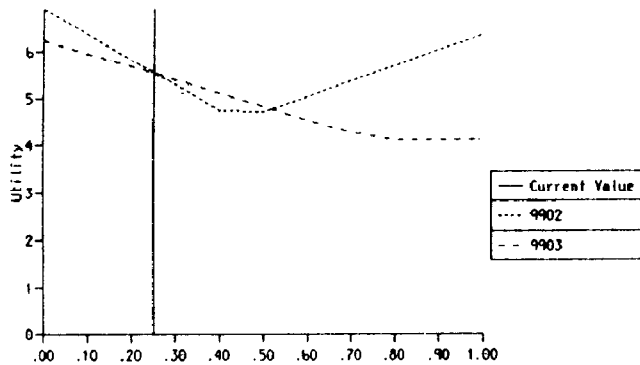
- Many of the input figures are suppositional - they came from peoples judgements
- What matters are trends - in what direction do preferences alter as parameters change and is it a lot or a little
- Are levels of preference large or small

A byproduct of this was that it was somewhat refreshing to see graphs which didn't have all their text in a mixture of proportionally spaced Olde English Gothic and Triplex Italian.

Again - an idea which caused us to gain insight, what we were hoping for was that we could avoid the 'micrometer syndrome' of having people take finicky measurements from a very shifty basis of input judgements. Sadly, users felt uneasy about these unadorned graphs - we reverted to using ICU. But - importantly - we know why we chose to do what we did and the exploration didn't consume much time.

Typical of the graphs being produced is the one below, where we are examining (in the context of the opening tree example) whether to proceed or not from the viewpoint of varying the weight given to the second scoring attribute.

## Autovary of WEIGHTS(2)



## CONCLUSIONS

- a) Some of the traditional APL market has moved away, attracted by easy-to-use tools for straightforward problems; as the limitations of these tools become apparent to their users APL is in a strong position to reassert its advantages.
- b) The implementation path followed by this development has been rather different from that which would have been taken using a language like Fortran - imitation never was a strength of APL.
- c) Many end users have been reluctant to get involved with APL because of past all-or-nothing commitment. An approach of 'only what you need', or even of not saying that the notation is APL helps overcome this problem.
- d) Notational styles like direct definition and vector notation are both palatable to the newcomer and have a natural feel.
- e) Evolving the system in the manner outlined above emphasised needed feature first before widening the useability.
- f) APL, particularly in the mainframe environment, has access to many rich resources; by exploiting this in addition to the powerful native syntax we can develop systems with powerful capabilities and attractive user interface.
- g) The system described above is computationally quite trivial, most of its complexity is within areas like user interface - the parts which had already been written and were imported intact.
- h) We got a lot of insight into contentious areas and were able to make our own choice of solution - not always the same as that of 'conventional wisdom'.
- i) This application is a niche which hasn't been filled by 'standard packages' - one role for APL in the future is to fill such niches

quickly - if only as a yardstick for evaluating packaged solutions as they appear.

## REFERENCES

- [1] Iverson, K.E., Programming Style In APL, An APL Users Meeting, Toronto 1978
- [2] Metzger, R.C., Extended Direct Definition of APL Functions, APL80, 1980
- [3] IBM Corporation, APL2 Programming: Using the Supplied Workspaces (SH20-9233), 1985
- [4] Gibson, L., Designing User-friendly APL Systems, 1982 APL Users Meeting, Toronto 1982