TK Solver Case Study: Performance Issues

This article presents several issues concerning minimizing solution times. For comparison purposes, our test computer was running at 450 MHz.

- **Display Intermediate Values**

A commonly overlooked item for new users is the "Display intermediate values" (DIV) setting in the Options menu. There are very few circumstances which might suggest having this box checked. The performance drag is significant for models which generate lots of list elements.

The TK Solver Library includes a nonlinear curve-fitting routine. If we run the routine with the sample data provided, the solution takes 0.096 seconds with DIV off. With DIV on, the solution time balloons to 3.971 seconds – slower by a factor of about 40.

- **Lots of Lists**

Be careful when working with matrices (multidimensional arrays). TK performs much better when working with a few long lists than it does with lots of short lists. The procedure below can be used to generate a matrix with 2000 rows (list elements) and 5 columns (lists). On our test computer, the solution time was 0.162 seconds.

Statement
for i = 1 to 5
for j = 1 to 2000
'x[i][j] = i/j
next j
next i

A very similar procedure can be used to generate the transpose of this matrix – that is, a matrix with 2000 columns (lists) and 5 rows (list elements). The same number of calculations are performed as before but TK is forced to account for more objects (list names) and this accounting time adds up.

Statement
for i = 1 to 5
for j = 1 to 2000
'x[j][i] = i/j
next j
next i

The variables i and j within the brackets are simply swapped. When this procedure is run, the solution time is 81.3 seconds – over 500 times slower!

- **Iterative Solver Accelerator**

The Iterative Solver Accelerator (ISA) option, when checked, avoids repeating unnecessary computations when iterating (that is, when guessing) for the solution of a model.

For example, in the following model it is necessary to compute the Fibonacci number of a variable and then use that result in a rule with more than one unknown. This would formerly have required the repetition of the first rule as many times as the second rule, since iteration was accomplished by the repetition of the model as a whole – even though the value of "v" would have been the same from one iteration to the next.

Here is the recursive procedure function Fibo, with input n and output F:

Statement
if n<3 then F = 1 else F = Fibo(n-1) + Fibo(n-2) + Fibo(n-3)

Here are the rules:

Rule
v = mod(Fibo(n),pi())
sin(x*v) = cos(x*v)

Checking the ISA, the first rule is executed only once and the result is then used over and over throughout the iteration process demanded by the second rule. With n input as 16 and using an initial guess of 0.1 for x, TK finds the solution shown below.

St	Input	Name	Output	Unit	Comment
		v	2.292669763		
	16	n			
		x	.3425692509		

This example (designed to be dramatic by using a time-consuming function) took 1.402 seconds without the accelerator and 0.142 seconds with the accelerator checked.

By default, the ISA is set to "Off" because there are rare cases involving the passing of list elements to and from rules in which not repeating rules during iteration will lead to different results. This occurs because of the sequence in which the rules are processed on multiple passes through the rule sheet. If a variable gets its value from a list element and the list element does not exist yet on the first pass through the rule sheet, the ISA option will ignore that variable on the second pass. The TK Help utility shows an example in which this will occur. If you are unsure if this is a cause for concern in one of your models, just try checking the ISA box and see if your solutions change.

- **Random Number Generation and Simulation**

The random number generation routines in the TK Library work most efficiently if they are used to generate values in bulk instead of one value at a time as is done in other languages or applications such as Microsoft Excel. The maximum number of values in a TK list is 32000. If your simulation requires more than 32000 random numbers, you can use several lists.

Here is a sample function call which generates a list, x, with 30000 values.

Statement
call random(0,30000,'x) ; generate random numbers

The computation time was 0.429 seconds.

A procedure function can then use these values one-by-one as it loops through a series of statements. As an example, the following TK procedure can be used to simulate the NBA draft lottery 10000 times.

```
Statement
y = 0
call random(0,30000,'x)    ; generate random numbers
k = 1    ; start random number counter
newyear:
call blank('fd)
call listcopy('f0,'fr)  ; initial teams balls
call listcopy('g0,'gd)  ; initial cumulative probabilities
call listcopy('fd0,'fd)
N = sum('f0)
y = y + 1    ; update year counter
call statmsg("Simulating Year #",y)
b = 1   ; start ball counter for year
pick:
call listcopy('fd,'hd)  ; set up temporary list
; pick a ball
t = g('x[k])   ; select team
N = N - f(t)  ; remove the winner's balls!
'T[b][y] = t   ; place selection in table
; update lists and functions with remaining teams
k = k + 1  ; update random number counter
tmin = min('fd)  ; lowest rank team remaining
call blank('fd)   ; blank the team list
call blank('gd)  ; blank the cumulative probability list
call listcopy('fr,'tfr)  ; set up temporary list
call blank('fr)   ; blank the team ball numbers list
v = 0   ; initialize counter
'gd[1] = 0
for d = 1 to 14-b   ; loop through the remaining teams to create the new functions
  hdd = 'hd[d]
  if t=hdd then continue
  v = v+1
  'fd[v] = 'hd[d]   ; team update
  'fr[v] = 'tfr[d]  ; team balls update
  'gd[v+1] = 'gd[v] + 'fr[v]/N   ; cumulative probability update
next d
b = b + 1  ; update ball counter
if b < 4 then goto repeat
for j = 4 to 13
  'T[j][y] = 'fd[j-3]
next j
call blank('hd)
call blank('tfr)
if k>30000 then goto stats
goto newyear
repeat:
call blank('hd)
call blank('tfr)
if k>30000 then goto stats
goto pick

stats:    ; count up the number of times each team got each pick
for j = 1 to 13
  for m = 1 to 13
    'S[j][m] = 0
  next m
next j
for i = 1 to 10000
  for n = 1 to 13
    'S['T[n][i]][n] = 'S['T[n][i]][n] + 1
  next n
next i
```