A Primer on Implementing Compressed Simulated Annealing for the Optimisation of a Constrained Simulation Model in Microsoft Excel®

Graeme J. Doole

28 June 2007
Working Paper 0701
School of Agricultural and Resource Economics
http://www.are.uwa.edu.au/

THE UNIVERSITY OF
WESTERN AUSTRALIA

Doole, G.J. (2007). *A primer on implementing compressed simulated annealing for the optimisation of a constrained simulation model in Microsoft Excel*®, Agricultural and Resource Economics Working Paper 0701, School of Agricultural and Resource Economics, University of Western Australia, Crawley, Australia.

Abstract. This short paper provides a simple introduction on how a simulation model implemented in Microsoft Excel® can be optimised using Visual Basic for Applications (VBA) programming and the compressed simulated annealing algorithm (Ohlmann et al., 2004; Ohlmann and Thomas, 2007). The standard simulated annealing procedure enters as a special case. Practical advice for determining the parameters that guide the stochastic search process in an annealing algorithm is also given.

Keywords. Compressed annealing, Microsoft Excel®, simulated annealing, simulation.

1. Introduction

This document describes how compressed simulated annealing (CSA) (Ohlmann et al., 2004; Ohlmann and Thomas, 2007) may be implemented in Microsoft Excel® and Visual Basic for Applications (VBA). Good introductions to programming in VBA are available in the Microsoft Excel® help section, Cottingham (1999), and Walkenbach (2004). The programming necessary to implement the following algorithm is very general and straightforward to implement, which aids the use of CSA for applied economic analysis. The description that follows uses an application to a complex weed management model, the RIM model (Pannell et al., 2004), for demonstration purposes. It follows the description of CSA in the paper "Optimisation of a large, constrained simulation model using compressed annealing: the value of French serradella (*Ornithopus sativus* Brot. cv. Cadiz) to Western Australian grain farming systems".

This code should act as a good template for analyses that wish to use compressed simulated annealing in an Excel® environment. The complete code without detailed documentation is presented in Appendix 1. This gives an overview of the structure of the CSA procedure without interruption by the numerous lines of comment provided in Sections 3 and 4.

2. Initial construction

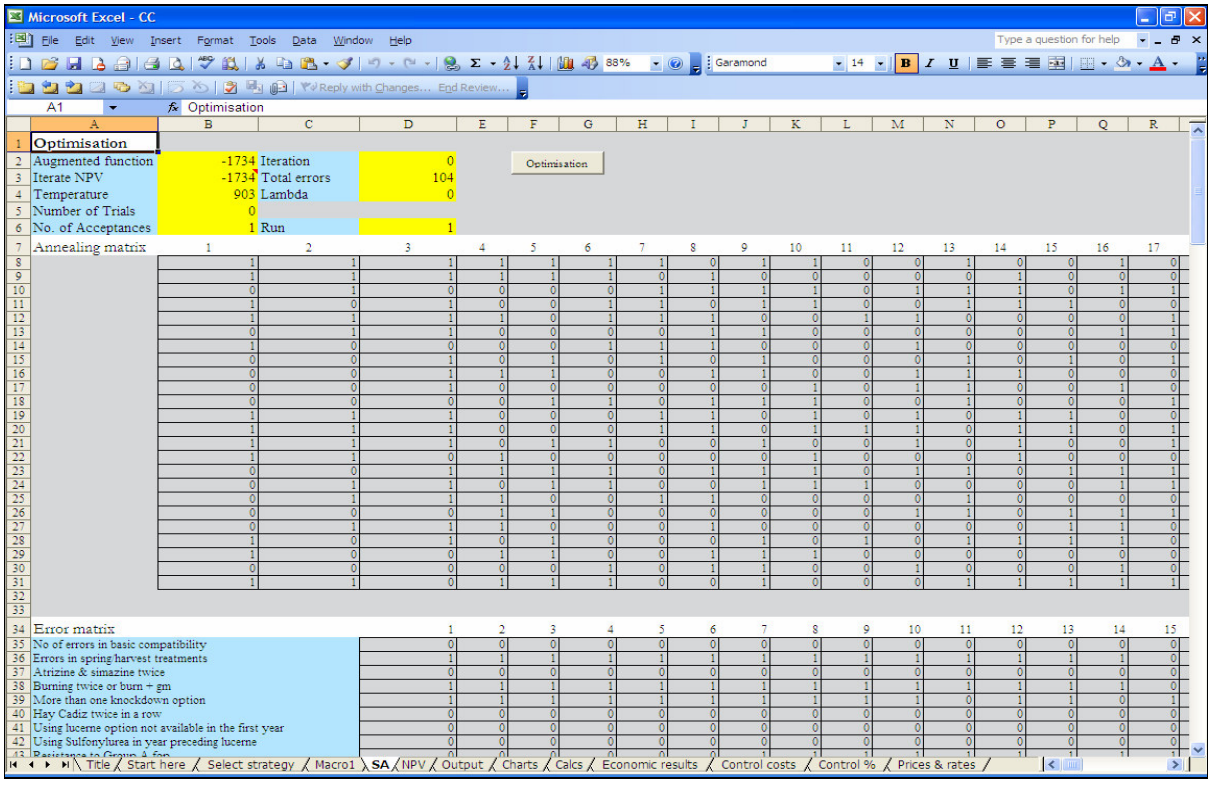
This presentation assumes that a spreadsheet model is constructed that possesses: (1) an array of discrete decision parameters presented in an array, (2) a mechanism to calculate an associated level of profit, and (3) a mechanism to calculate an associated number of infeasibilities. Factors (2) and (3) can usually be generated easily given a change in (1) as a spreadsheet can automatically recalculate these cells following one of the cells containing the decision parameters being updated.

First of all, insert a sheet called "SA" in the workbook containing the simulation model. ("SA" simply denotes "simulated annealing".) Construct the nine status boxes in yellow and their associated label cells as depicted in Figure 1. These labels denote the augmented functional, the net present value (NPV) calculated for each iteration, the temperature, the number of trials performed in this iteration, the number of acceptances in this iteration, the iteration number, the number of total errors associated with the existing configuration, the value of the penalty factor (i.e., lambda), and the run number. All of these numbers, except the NPV and the total number of errors, are computed in the algorithm and printed onto the worksheet during the solution of the model. They denote progress in the algorithm. This information is valuable when performing standard runs and to

improve understanding of the behaviour of a given model when annealing parameters are being "tuned".

Construct the annealing matrix containing the array of binary weed treatments (i.e., decision parameters) (Figure 1). In the RIM model, this matrix has 20 columns and 24 rows. The twenty columns represent the planning horizon incorporated in the model, and the number of rows is the maximum number of treatments available in each year. This yields 20*24=480 cells denoting binary weed treatments. These interact with the simulation model (described on the other sheets in the model) that calculates the NPV (Cell B3, labelled the *Iterate NPV* on the "SA" sheet) and number of errors (Cell D3, labelled the *Total Errors* on the "SA" sheet) associated with each configuration. The error matrix (see bottom of Figure 1) calculates the source of infeasibilities in each year to aid diagnostic work, particularly during sensitivity analysis when different configurations of the model parameters may markedly alter the nature of the optimal solution.

Figure 1. Screenshot of the SA sheet in the RIM model.



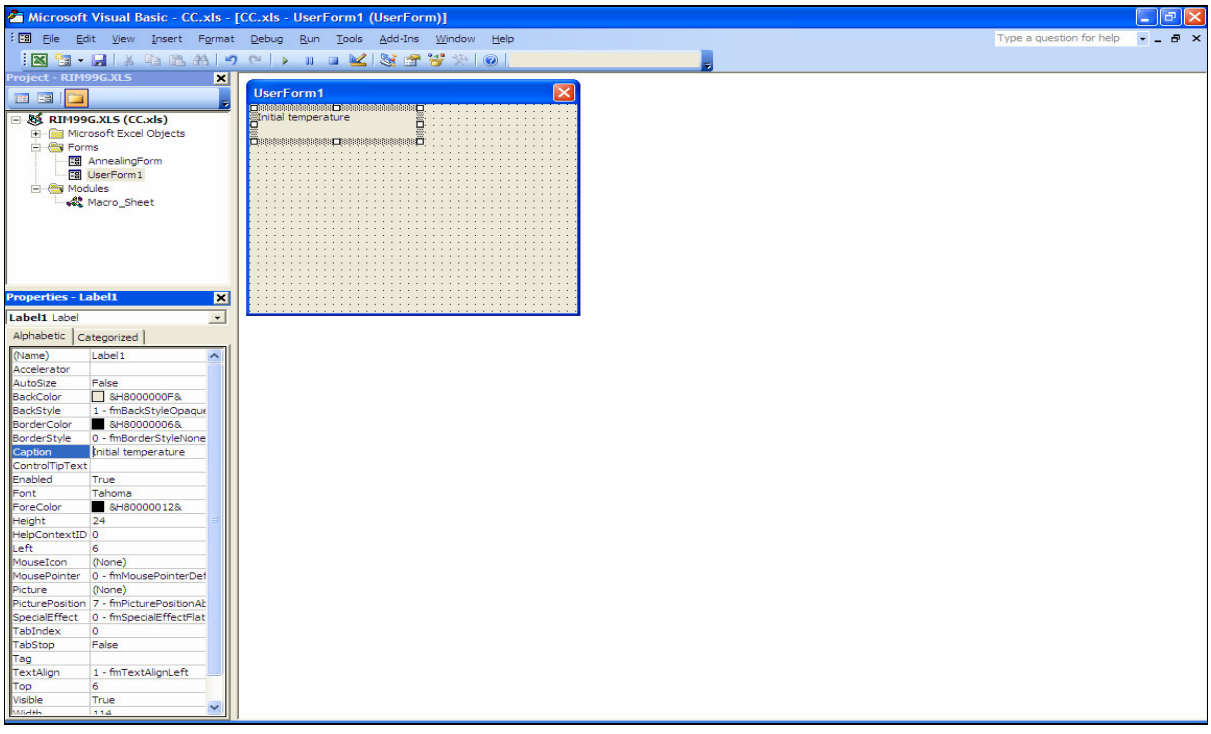
3. Programming in VBA

This section describes the VBA programming required to implement the CSA algorithm.

Click on **Tools/Macro/Visual Basic Editor**. Click **Insert/UserForm**. Add label to the form by clicking the **A** button on the Toolbox bar and sizing it appropriately on the UserForm (Figure 2). Click on **Caption** in the Properties box in the bottom left hand side (LHS) of the screen and type in *Initial temperature* (Figure 2). Repeat this exercise of creating a text label and changing its caption for *length of Markov chain*, *decompression rate*, *decrement in temperature*, *maximum compression parameter*, *acceptance rate*, and *number of runs* (see alignment in Figure 3).

Add Textbox by clicking the **ab** button on the Toolbox bar and sizing it appropriately on the UserForm. Change textbox name to *txtIT* by changing top box (i.e., (Name)) in Properties box in bottom left hand corner of the screen (Figure 2). Repeat this process for the parameters: length of Markov chain (name box *txtLMC*) (this is the maximum number of trials that may be performed in each inner loop), decompression rate (name box *txtDR*), decrement in temperature (name box *txtDT*), maximum penalty factor (name box *txtML*), acceptance rate (name box *txtAR*), and number of runs (name box *txtNR*). These names, e.g. *txtML*, will be used to pass values from the UserForm into the algorithm.

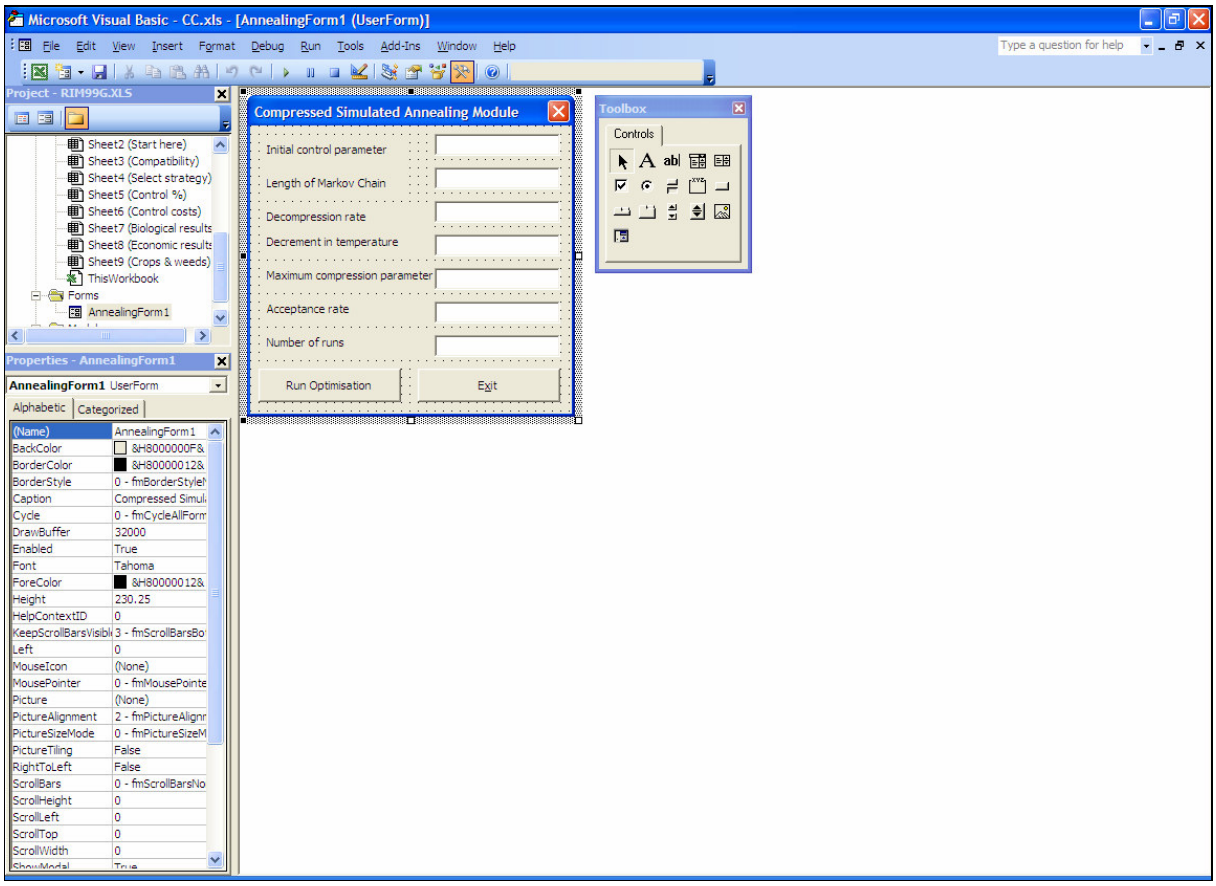
Figure 2. Adding a label to a UserForm.



Highlight the UserForm name on the Project screen by clicking on it. Replace name of the UserForm in the Properties box with *AnnealingForm1*. Add caption in Properties box as *Compressed Simulated Annealing Module* (Figure 3). Add command button by clicking the

appropriate tool in the Toolbox bar (it resembles a shaded rectangle) and locating it on the bottom LHS of the UserForm (Figure 3). Add (Name) in the Properties section. Call this *cmdOptimisation*. Change caption in Properties to be *Run optimisation*. Add a second command button by clicking the appropriate tool in the Toolbox bar and locating it on the bottom RHS of the UserForm (Figure 3). Add (Name) in Properties. Call it *cmdExit*. Add caption in Properties as *Exit*. The completed interface of the UserForm is shown in Figure 3.

Figure 3. Completed interface of the UserForm.



Double click on empty space on UserForm. The existing code should read:

Private Sub cmdExit_Click()
End
End Sub
Private Sub Optimisation_Click()
Close

Delete any other text in the code (including the *End Sub* command that normally follows *Close* on the fifth line) window that may have been generated accidentally while doing the activities above. (This *End Sub* command is required at the end of the *completed* code, but the complete code is of a

decent length, so it is easier to delete this command and type it in later when instructed.) The first three lines listed above stop the algorithm when the **Exit** button in the UserForm is pressed. The next two begin optimisation using the simulated annealing algorithm once the **Run optimisation** button on the UserForm is depressed.

(In the VBA language, the symbol ‘ denotes a text instruction and is not compiled as an instruction to the computer. These are therefore used extensively in the following to document the code in greater detail.)

Next, type the following underneath the *Close* command on the fifth line of the existing code:

```
‘Enter data from UserForm
Dim IT As Double, LMC, DR, DT, ML, AR, NR
IT = txtIT.Text
LMC = txtLMC.Text
DR = txtDR.Text
DT = txtDT.Text
ML = txtML.Text
AR = txtAR.Text
NR = txtNR.Text
```

This enters data from the UserForm and gives each parameter a name for the computer to call in the code. Also, initiate other variables by typing:

```
Dim R as Double, POWER, Pacc, Temp
```

State that the number of completed CSA runs is to be less than a maximum number in case of typing error and to reduce computational demand. (However, this upper limit may be removed depending on the application and the computational resources available.) Also, a loop is initiated so multiple annealing runs may be performed given the large size of some problems and the stochastic nature of the search algorithm. These features are introduced by typing:

```
'Maximum number of annealing runs is 50
If NR > 50 Then
NR = 50
End If
R = 1
While R < NR + 1
Sheets("SA").Cells(6, 4) = R ‘print run number to worksheet
```

The loop initiated in the last two lines is completed at the bottom of the documented code (see below) given that each run contains a complete implementation of the CSA algorithm.

Add additional text:

```
Dim NPVb(500) As Integer 'define matrix that measures if solution has converged
Temp = IT ‘define initial temperature
Sheets("SA").Cells(4, 2) = Temp ‘print temperature to worksheet
```

The annealing matrix containing binary numbers for each treatment in every year is next constructed in memory. There are 480 elements in this array as described in Section 2 above. The array is constructed through the commands:

```
Sheets("SA").Activate ‘activate the sheet
Dim SAM(480) As Integer ‘construct string representing matrix
For J = 1 To 480 ‘fill SAM matrix with random binary numbers
Randomize ‘this command initialises the random number generator by giving it a new seed value
SAM(J) = Int(2 * Rnd)
Next
'Print matrix to worksheet
For J = 1 To 480
ColM = 1 + (J Mod 20) ‘this command allows the string to be printed as a matrix to the worksheet
‘it is Mod 20 as there are twenty columns (i.e., years) in the annealing matrix (see Figure 1)
If ColM = 1 Then
ColM = 21 'this loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((J - 1) \ 20), ColM) = SAM(J)
Next
```

Define parameters for annealing process:

```
Dim NPVo As Single, NPVn, TError, NA, NT, L
L=0 ‘initial lambda for penalty function is zero
Sheets("SA").Cells(4, 4) = L ‘print lambda to sheet
I = 0 ‘iteration number
Sheets("SA").Cells(2, 4) = I ‘print iteration number to sheet
```

Obtain NPV associated with the initial configuration:

```
NPVo = Sheets("SA").Cells(3, 2) 'get NPV value from worksheet
TError = Sheets("SA").Cells(3, 4) 'get number of errors from worksheet
AFo = NPVo - TError * L 'calculate augmented functional
```

Next, construct the loops for annealing:

```
Do Until Temp < 0.0001 ‘run outer loop until temperature is less than epsilon
NA = 0 ‘initialise number of acceptances
NT = 0 ‘initialise number of trials
Do
Sheets("SA").Cells(5, 2) = NT ‘print to worksheet
Randomize 'these lines of code randomly select a cell for updating
CurCell = Int((480 * Rnd) + 1)
‘Swap its binary value
If SAM(CurCell) = 0 Then
SAM(CurCell) = 1
Else
SAM(CurCell) = 0
End If
ColM = 1 + (CurCell Mod 20) 'print updated cell to worksheet
If ColM = 1 Then
ColM = 21 'this loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((CurCell - 1) \ 20), ColM) = SAM(CurCell)
NPVn = Sheets("SA").Cells(3, 2) 'get profit associated with new configuration from worksheet
TError = Sheets("SA").Cells(3, 4) 'also get number of errors associated with new solution from
worksheet
AFn = NPVn - TError * L ‘calculate new augmented functional
If AFn >= AFo Then 'always accept higher augmented profit
AFo = AFn
Sheets("SA").Cells(2, 2) = AFo
NA = NA + 1 ‘number of acceptances is increased by one
Sheets("SA").Cells(6, 2) = NA ‘print number of acceptances to worksheet
Else
POWER = (AFn - AFo) / Temp ‘if augmented profit is not higher, construct Boltzmann acceptance
criterion
Pacc = Exp(POWER)
End If
Randomize
If Pacc > Rnd Then ‘test solutions using stochastic acceptance function of annealing theory
AFo = AFn  'update iterates
Sheets("SA").Cells(2, 2) = AFo
NA = NA + 1
Sheets("SA").Cells(6, 2) = NA
Else 'undo change
If SAM(CurCell) = 1 Then
SAM(CurCell) = 0
Else
SAM(CurCell) = 1
End If
‘Print updated cell to worksheet
ColM = 1 + (CurCell Mod 20)
If ColM = 1 Then
ColM = 21 'This loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((CurCell - 1) \ 20), ColM) = SAM(CurCell)
End If
End If
```

```
If NA = Int(LMC * AR) Then Exit Do 'break inner loop if number of acceptances reaches upper bound
If NT = LMC Then Exit Do 'break inner loop if number of trials reaches upper bound
NT = NT + 1 'otherwise increase number of trials by one and begin inner loop again
Loop
```

(Note: Depending on the magnitude of the objective function, scaling may be required in the model to prevent overflow when *Pacc* is calculated.)

The following text is useful to include in the inner loop to allow the Exit button on the UserForm to stop the annealing algorithm quickly.

```
A = Timer
While Timer < A + 0.005
  DoEvents
Wend
```

The end of the outer loop is continued:

```
NPVb(I) = AFo 'enter profit term into convergence matrix
If I > 10 Then 'break loop if augmented profit has converged
If NPVb(I - 8) = NPVb(I - 9) Then
If NPVb(I - 7) = NPVb(I - 8) Then
If NPVb(I - 6) = NPVb(I - 7) Then
If NPVb(I - 5) = NPVb(I - 6) Then
If NPVb(I - 4) = NPVb(I - 5) Then
If NPVb(I - 3) = NPVb(I - 4) Then
If NPVb(I - 2) = NPVb(I - 3) Then
If NPVb(I - 1) = NPVb(I - 2) Then
If NPVb(I) = NPVb(I - 1) Then Exit Do
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
I = I + 1 'increase iteration count by one
L = ML * (1 - Exp(-DL * I)) 'update penalty factor
Temp = Temp * DT 'reduce temperature via cooling schedule
Sheets("SA").Cells(2, 4) = I 'print updated values to worksheet
Sheets("SA").Cells(4, 2) = Temp
Sheets("SA").Cells(6, 2) = 0
Sheets("SA").Cells(4, 4) = L
Loop
```

Solution and iteration results for each converged annealing run are printed to an "NPV" sheet for post-run analysis.

```
Sheets("NPV").Activate
Sheets("NPV").Cells(R + 1, 2) = AFo
Sheets("NPV").Cells(R + 1, 3) = TError
Sheets("NPV").Cells(R + 1, 4) = L
Sheets("NPV").Cells(R + 1, 5) = I
```

Also, the optimal configuration of treatments for each converged annealing run is printed to an "Output" sheet for post-run analysis.

```
Sheets("Output").Activate
'Print matrix to worksheet
For J = 1 To 480
ColM = 1 + (J Mod 20)
If ColM = 1 Then
ColM = 21 'this loop corrects an error associated with use of Mod function
End If
Sheets("Output").Cells((25 * (R - 1) + 2) + ((J - 1) \ 20), ColM) = SAM(J)
Next
R = R + 1 'this updates the run counter, the algorithm subsequently starts the next run at the top of
the programme
Wend
Close
End Sub
```

It is useful to type the following at the end of the code. This defines starting values for the text boxes on the UserForm that users may change as desired.

```
Private Sub UserForm_Activate()
txtIT.Text = 903
txtLMC.Text = 480
txtDR.Text = 0.04
txtDT.Text = 0.925
txtML.Text = 413
txtAR.Text = 0.5
txtNR.Text = 1
End Sub
```

4. Creating a command button to open UserForm

Move back into the Excel spreadsheet, activate SA sheet, and click **View/Toolbars/Control Toolbox**. Draw a *command button* on a cell using the appropriate button on this toolbox (the button resembles a shaded rectangle) and defining its position on the form. Right click the command button and edit its name to "Optimisation". Click off the command button, then double click on it.

This will open the VBA editor. Enter the middle line from the following, so the code reads on completion:

Private Sub CommandButton1_Click()
AnnealingForm1.Show
End Sub

This ensures that pressing this command button opens the annealing form.

5. Formulation of the parameters for the CSA procedure

This section describes in detail how the annealing parameters were estimated for the application of CSA described in the associated application.

5.1 Neighbourhood generation mechanism

The optimisation process treats each admissible weed treatment as a member of a string of length $\mu = 20 \cdot j_{\max}$, where j_{max} is the number of weed treatments available in each year over the planning horizon. (There are $\mu = 20 \cdot 24 = 480$ members of the string in this application.) Here, $u_\iota = \{0,1\}$ where $\iota = [1,2,...,\mu]$, with $u_\iota = 1$ representing the use of a treatment and $u_\iota = 0$ representing no use. Each trial consists of:

1. generating a random number $\beta = \mu\eta$ (where η is a uniformly-distributed random variable on [0,1]) denoting the index of the decision parameter to change; and

2. performing the update $u_\iota^{new} = 1 - u_\iota^{old}$, where $\iota = \beta$ and the superscripts *old* and *new* denote the value of the binary decision variable before and after the update respectively.

5.2 Initial temperature

The initial value of the temperature is selected so that almost all transitions are accepted. A random walk over 5000 configurations was used to compute the mean absolute change in the objective function ($|\Delta J_m|$). The initial level of the temperature was then selected through the rule,

$$K_0 = \frac{|\Delta J_m|}{\ln(\Xi^{-1})}, \tag{1}$$

where Ξ is an appropriate acceptance ratio (the ratio of the number of acceptances to the number of trials). A ratio (Ξ) of 0.99 is selected to ensure that an adequate search is undertaken at the initial temperature. An initial temperature of $K_0 = 903$ was identified using this process. This method of determining the initial temperature is standard practice in annealing theory (van Laarhoven, 1988; Aarts and Korst, 1989). The following code may aid in the construction of a programme to identify $\left|\Delta J_m\right|$ for an Excel® model. This code is based heavily upon the above description of programming annealing models in VBA, representing the efficiency with which such a programme may be formulated once the initial code is written for a given problem. To construct the UserForm for this programme, simply adapt the instructions above in Section 2 accordingly.

```
Private Sub cmdExit_Click()
End
End Sub
Private Sub cmdOptimisation_Click()
Close
'Enter the parameter from the UserForm
Dim LRW As Double 'the UserForm now only has one cell denoting the length of the random walk
LRW = txtLRW.Text
'Construct random binary annealing matrix in memory
Sheets("SA").Activate
Dim SAM(480) As Integer
For J = 1 To 480
Randomize
SAM(J) = Int(2 * Rnd)
Next
'Print matrix to worksheet
For J = 1 To 480
ColM = 1 + (J Mod 20)
If ColM = 1 Then
ColM = 21 'this loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((J - 1) \ 20), ColM) = SAM(J)
Next
Workbooks("Randomwalk.xls").Activate
'Get NPV from sheet
Dim C0 As Single, C1, CA
C1 = Sheets("SA").Cells(3, 2) 'get NPV from sheet
For J = 1 To LRW
C0 = C1
'Select cell for updating
Randomize
CurCell = Int((480 * Rnd) + 1)
'Swap its binary value
If SAM(CurCell) = 0 Then
SAM(CurCell) = 1
Else
SAM(CurCell) = 0
```

```
End If
'Print updated cell to worksheet
ColM = 1 + (CurCell Mod 20)
If ColM = 1 Then
ColM = 21 'This loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((CurCell - 1) \ 20), ColM) = SAM(CurCell)
'Get current profit from worksheet
C1 = Sheets("SA").Cells(3, 2)
A = Timer
While Timer < A + 0.005
DoEvents
Wend
CA = abs(C0 - C1)
Sheets("SA").Cells(55 + J, 1) = CA 'absolute difference in objective function values are printed to
the sheet, the mean can then be determined on the spreadsheet
J = J + 1
Sheets("SA").Cells(2, 2) = J
Next
Close
End Sub
```

5.3 Temperature-decrement rule

The temperature is decreased according to the rule $K_{t+1} = \sigma K_t$, where $\sigma = 0.925$. This value is determined through extensive experimentation with the model since it is strongly related to the other components of the cooling and compression schedules. Common values are between $\sigma = 0.88$ and $\sigma = 0.99$, so experimentation with values between these bounds usually identifies a suitable candidate (Aarts and Korst, 1989).

5.4 Maximum number of trials at each temperature

A certain number of acceptances should occur at each level of the temperature to establish stability of the solution. (Such quasi-equilibrium is necessary to theoretically obtain the global optimum through simulated annealing (Hajek, 1988; Ohlmann et al., 2004).) However, $NT \to \infty$ as $K \to 0$ if a given number of acceptances is required, so a maximum number of trials is defined ($NTMAX = 480$) for a given K (Aarts and Korst, 1989). This value of $NTMAX$ is the largest string length encountered over the evaluated rotations and thus ensures, at least theoretically, that each treatment is visited at each level of the temperature (Kirkpatrick et al., 1983; Johnson et al., 1989, 1991). An inner loop is terminated if this upper bound is met or if the number of acceptances reaches 50 percent of $NTMAX$. The acceptance threshold helps to minimise computational effort at high temperatures, while providing for the establishment of quasi-equilibrium (van Laarhoven and Aarts, 1987). This parameter value is determined through experimentation with the model.

5.5 Maximum penalty value

The maximum penalty rate is determined by a process based on that of Ohlmann and Thomas (2007). This technique is implemented here through conducting a random walk over 5000 configurations and evaluating,

$$\hat{\lambda} = \max\left\{\frac{\pi(u_{j,t})}{\xi(u_{j,t})}\frac{\kappa}{(1-\kappa)}\right\}, \tag{2}$$

where $\pi(u_{j,t})$ is net present value as a function of the configuration of the decision parameters, $\xi(u_{j,t})$ is the number of infeasibilities present in the model as a function of the configuration of the decision parameters, and κ is a pre-defined proportion of the objective function composed of the penalty term at $\lambda = \hat{\lambda}$. A value of $\kappa = 0.95$ is selected since a penalty schedule should evolve towards a large maximum value to significantly penalise infeasible transitions as the annealing algorithm converges. The corresponding value is $\hat{\lambda} = 413$. This process is implemented using a variant of the code given in Section 5.2, the primary difference being that the total errors must also enter the code from the worksheet to compute (2).

5.6 Penalty rate

The penalty rate ($\gamma = 0.04$) is selected through extensive experimentation with the model.

5.7 Termination condition

The algorithm is stopped at $K = 0.0001$ or when the terminal configuration has not changed for ten consecutive bands of the temperature (i.e., ten repetitions of the outer loop). This stability of the solution is taken to signify convergence and is achieved as $K \to 0$, except where incorrect parameterisation of the penalty function severely limits the breadth of the search, which occurred occasionally during the tuning of the parameters for the cooling and compression schedules.

References

Aarts, E.H.L. and Korst, J. (1989), *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*, Wiley, New York.

Cottingham, M. (1999), *Excel 2000 Developer's Handbook*, Sybex, San Francisco.

Hajek, B. (1988), 'Cooling schedules for optimal annealing', *Mathematics of Operations Research* 13, pp. 311-29.

Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C. (1989), 'Optimization by simulated annealing: an experimental evaluation. Part I: graph partitioning', *Operations Research* 37, pp. 865-592.

Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C. (1991), 'Optimization by simulated annealing: an experimental evaluation; part II, graph colouring and number partitioning', *Operations Research* 39(3), pp. 378-406.

Kirkpatrick, S., Gelatt, G.C. and Vecchi, M.P. (1983), 'Optimization by simulated annealing', *Science* 220(4598), pp. 671-80.

Ohlmann, J.W., Bean, J.C. and Henderson, S.G. (2004), 'Convergence in probability of compressed annealing', *Mathematics of Operations Research* 29, pp. 837-60.

Ohlmann, J.W. and Thomas, B. (2007), 'A compressed annealing approach to the travelling salesman problem with time windows', *INFORMS Journal on Computing* 19, pp. 80-90.

Pannell, D.J., Stewart, V., Bennett, A., Monjardino, M., Schmidt, C. and Powles, S.B. (2004), 'RIM: a bioeconomic model for integrated weed management of *Lolium rigidum* in Western Australia', *Agricultural Systems* 79, pp. 305-25.

van Laarhoven, P.J.M. and Aarts, E.H.L. (1987), *Simulated annealing*, Reidel, Dordrecht.

van Laarhoven, P.J.M. (1988), *Theoretical and computational aspects of simulated annealing*, Centrum voor Wiskunde en Informatica, Amsterdam.

Walkenbach, J. (2004), *Excel 2003 power programming with VBA*, Wiley, New York.

Appendix 1. Completed compressed simulated annealing in VBA code.

The final code for the CSA algorithm should read:

```
Private Sub cmdExit_Click()
End
End Sub
Private Sub Optimisation_Click()
Close
'Enter data from UserForm
Dim IT As Double, LMC, DR, DT, ML, AR, NR
IT = txtIT.Text
LMC = txtLMC.Text
```

```
DR = txtDR.Text
DT = txtDT.Text
ML = txtML.Text
AR = txtAR.Text
NR = txtNR.Text
Dim R as Double, POWER, Pacc, Temp
'Maximum number of annealing runs is 50
If NR > 50 Then
NR = 50
End If
R = 1
While R < NR + 1
Sheets("SA").Cells(6, 4) = R
Dim NPVb(500) As Integer 'define matrix that measures if solution has converged
Temp = IT ‘define initial temperature
Sheets("SA").Cells(4, 2) = Temp ‘print temperature to worksheet
Sheets("SA").Activate ‘activate the sheet
Dim SAM(480) As Integer ‘construct string representing matrix
For J = 1 To 480 ‘fill SAM matrix with random binary numbers
Randomize ‘this command initialises the random number generator by giving it a new seed value
SAM(J) = Int(2 * Rnd)
Next
'Print matrix to worksheet
For J = 1 To 480
ColM = 1 + (J Mod 20) ‘this command allows the string to be printed as a matrix to the worksheet
If ColM = 1 Then
ColM = 21 'this loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((J - 1) \ 20), ColM) = SAM(J)
Next
Dim NPVo As Single, NPVn, TError, NA, NT, L
L=0 ‘initial lambda for penalty function is zero
Sheets("SA").Cells(4, 4) = L ‘print lambda to sheet
I = 0 ‘iteration number
Sheets("SA").Cells(2, 4) = I ‘print iteration number to sheet
NPVo = Sheets("SA").Cells(3, 2) 'get NPV value from worksheet
TError = Sheets("SA").Cells(3, 4) 'get number of errors from worksheet
AFo = NPVo - TError * L 'calculate augmented functional
Do Until Temp < 0.0001 ‘run outer loop until temperature is less than epsilon
NA = 0 ‘initialise number of acceptances
NT = 0 ‘initialise number of trials
Do
Sheets("SA").Cells(5, 2) = NT ‘print to worksheet
Randomize 'these lines of code randomly select a cell for updating
CurCell = Int((480 * Rnd) + 1)
‘Swap its binary value
If SAM(CurCell) = 0 Then
SAM(CurCell) = 1
Else
SAM(CurCell) = 0
End If
ColM = 1 + (CurCell Mod 20)    'print updated cell to worksheet
```

```
If ColM = 1 Then
ColM = 21 'This loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((CurCell - 1) \ 20), ColM) = SAM(CurCell)
NPVn = Sheets("SA").Cells(3, 2) 'get profit associated with new configuration from worksheet
TError = Sheets("SA").Cells(3, 4) 'also get number of errors associated with new solution from
worksheet
AFn = NPVn - TError * L ‘calculate new augmented functional
If AFn >= AFo Then 'always accept higher augmented profit
AFo = AFn
Sheets("SA").Cells(2, 2) = AFo
NA = NA + 1 ‘number of acceptances is increased by one
Sheets("SA").Cells(6, 2) = NA ‘print number of acceptances to worksheet
Else
POWER = (AFn - AFo) / Temp ‘if augmented profit is not higher, construct Boltzmann acceptance
criterion
Pacc = Exp(POWER)
End If
Randomize
If Pacc > Rnd Then ‘test solutions using stochastic acceptance function of annealing theory
AFo = AFn  'Update iterates
Sheets("SA").Cells(2, 2) = AFo
NA = NA + 1
Sheets("SA").Cells(6, 2) = NA
Else 'undo change
If SAM(CurCell) = 1 Then
SAM(CurCell) = 0
Else
SAM(CurCell) = 1
End If
‘Print updated cell to worksheet
ColM = 1 + (CurCell Mod 20)
If ColM = 1 Then
ColM = 21 'This loop corrects an error associated with use of Mod function
End If
Sheets("SA").Cells(8 + ((CurCell - 1) \ 20), ColM) = SAM(CurCell)
End If
End If
If NA = Int(LMC * AR) Then Exit Do ‘break inner loop if number of acceptances reaches upper
bound
If NT = 480 Then Exit Do ‘break inner loop if number of trials reaches upper bound
NT = NT + 1 ‘otherwise increase number of trials by one and begin inner loop again
A = Timer
While Timer < A + 0.005
  DoEvents
Wend
Loop
NPVb(I) = AFo ‘enter profit term into convergence matrix
If I > 10 Then 'break loop if augmented profit has converged
If NPVb(I - 8) = NPVb(I - 9) Then
If NPVb(I - 7) = NPVb(I - 8) Then
If NPVb(I - 6) = NPVb(I - 7) Then
```

```
If NPVb(I - 5) = NPVb(I - 6) Then
If NPVb(I - 4) = NPVb(I - 5) Then
If NPVb(I - 3) = NPVb(I - 4) Then
If NPVb(I - 2) = NPVb(I - 3) Then
If NPVb(I - 1) = NPVb(I - 2) Then
If NPVb(I) = NPVb(I - 1) Then Exit Do
End If
End If
End If
End If
End If
End If
End If
End If
End If
I = I + 1 ‘increase iteration count by one
L = ML * (1 - Exp(-DL * I)) ‘update penalty factor
Temp = Temp * DT
Sheets("SA").Cells(2, 4) = I ‘print updated values to worksheet
Sheets("SA").Cells(4, 2) = Temp
Sheets("SA").Cells(6, 2) = 0
Sheets("SA").Cells(4, 4) = L
Loop
Sheets("NPV").Activate
Sheets("NPV").Cells(R + 1, 2) = AFo
Sheets("NPV").Cells(R + 1, 3) = TError
Sheets("NPV").Cells(R + 1, 4) = L
Sheets("NPV").Cells(R + 1, 5) = I
Sheets("Output").Activate
'Print matrix to worksheet
For J = 1 To 480
ColM = 1 + (J Mod 20)
If ColM = 1 Then
ColM = 21 'This loop corrects an error associated with use of Mod function
End If
Sheets("Output").Cells((25 * (R - 1) + 2) + ((J - 1) \ 20), ColM) = SAM(J)
Next
R = R + 1 ‘this updates the run counter, the algorithm subsequently starts at the next run at top of
sheet
Wend
Close
End Sub

Private Sub UserForm_Activate()
txtIT.Text = 903
txtLMC.Text = 480
txtDR.Text = 0.04
txtDT.Text = 0.925
txtML.Text = 413
txtAR.Text = 0.5
txtNR.Text = 1
End Sub
```