Appendix A

BasicGA: Code for Genetic Algorithms

Note April 8, 1998: This listing is only approximate. I have a running version of the code in the Excel98 file gavsolexcelf98.xls, which is the solution to the programming case for DOPIM 101, spring 1996.

A.1 Introduction

The purpose of this appendix is to lay out and discuss the code for BasicGA. BasicGA is a program having some (very) basic genetic algorithm capabilities. It was written in Visual Basic (Microsoft) and works in the Visual Basic for Applications (VBA) dialect. BasicGA works in Excel. The purpose of BasicGA is to serve as a shell or starting point for developing applications using genetic algorithms, especially in a classroom environment. My intention, thus, has been to make BasicGA as implementation-independent as possible.

Note: For program development, debugging, and other purposes, I have often substituted a stub routine in BasicGA for what would be an actual routine in a full application. Such stubs will have “stub” appended to the actual name, sometimes prefixed and sometimes postfixed. For example, the stub for InitializeGA is InitializeGASStub but could be StubInitializeGA. Also, in this documentation, names of code objects, e.g., procedures and

\(^1\)A stub is a procedure that is intendedly not in final form, but is used during program development and testing.
variables, will be given in a typewriter font, as we have just seen with InitializeGA.

A.2 Declarations

The purpose of this section is to describe the declared, global or public, variables for BasicGA. Here follows the declarations section of the BasicGA code. It is written in Microsoft Visual Basic 3.0 and has also been tested in the Microsoft Excel 5.0 environment.

'sok 951127: This is in file: GACODE.BAS in 
'the folder clapopo3 
'Am freezing this for now. Call it the BasicGA program, 
'version 951127.

Option Explicit

'******************************************************************************
'*************** Below, constants declared that ***
'*************** should be read in. ***************

'sok 951126: Note: These program variables, 
'in a non-stubbed 
'environment, need to be declared in the declarations 
'section. They are so declared, but I have commented 
'out the declarations (see below).

'+--------------------------+
'+++++ from GetGARunPars ++++++

Const NumberOfGenerations = 20
'GetNumberOfGenerations
Const PopulationSize = 100
'GetPopulationSize
Const CrossoverRate = .77
'GetCrossoverRate
A.2. DECLARATIONS

Const MutationRate = .23
'GetMutationRate
Const bestNSaved = 100
'GetBestNSaved
'+++++++++++++++++++++++++++++++++++++++++++++++++++<<<<<<<
'++++++++++++ from GetModelRunPars +++++++++++++
Const NumberOfDecisionVariables = 4

'GetOutputSize
Const OutputSize = 2

'+++++++++++++++++++++++++++++++++++++++++++++++++++<<<<<<<
'+++++++++++++++++++++++++++++++++++++++++++++++++++<<<<<<<
'++++++++++++ from/for InitDVarInfo/StubInitDVarInfo +++++++

Dim DecisionVariableInfo(1 To NumberOfDecisionVariables,
=> 1 To 4) As Double

Const DecisionVariableInfo011 = 5  'r, low
Const DecisionVariableInfo012 = 20  'r, high
Const DecisionVariableInfo013 = 0  'r, not integer
Const DecisionVariableInfo014 = 0  'r, no grid search

Const DecisionVariableInfo021 = 10  'v, low
Const DecisionVariableInfo022 = 30  'v, high
Const DecisionVariableInfo023 = 0  'v, not integer
Const DecisionVariableInfo024 = 0  'v, no grid search

Const DecisionVariableInfo031 = 15  'u, low
Const DecisionVariableInfo032 = 25  'u, high
Const DecisionVariableInfo033 = 0  'u, not integer
Const DecisionVariableInfo034 = 0  'u, no grid search

Const DecisionVariableInfo041 = 200  'l, low
Const DecisionVariableInfo042 = 300  'l, high
Const DecisionVariableInfo043 = 0  'l, not integer
Const DecisionVariableInfo044 = 0  'l, no grid search
' ++++++++++++++++++++++++++++++++++++++++++++++
'******************************************************************************
'******************************************************************************
'************ Above, constants declared that ***
'************ should be read in. *************

' Global variables

'+++++ from GetGARunPars
' **** but explicitly declared above ++++++++++

' Dim NumberOfGenerations as Integer
' Dim PopulationSize as Integer
' Dim CrossoverRate As Double
' Dim MutationRate As Double
' Dim bestNSaved as Integer

' ++++++++++ from GetModelRunPars
' ++++++++++++but explicitly declared above +++++++++

' Dim NumberOfDecisionVariables As Integer
' Dim OutputSize As Integer

' +++++++++++++++++++++++++++++++++++++++++++++++++++++

Dim Index As Integer
Global CurrentGeneration() As Double
Global AbsoluteFitness() As Double
Dim ChromosomeCopySpace() As Double
Dim RelativeFitness() As Double
Dim CrossoverLikelihood() As Double
Dim BestNCurrentSaveSet() As Double
Dim LowestAbsoluteFitness As Double
Dim HighestAbsoluteFitness As Double
Dim CurrentIdNum As Double
Dim NumberOfGenerationsSoFar As Integer
Dim CrossoverPoint As Integer

Dim NoisyOutput As Integer ' 1 = show lots of output;
'0 = don’t

The general structure and plan for the program is simple. Everything revolves around two arrays.

First, the array CurrentGeneration holds the current generation of chromosomes, one chromosome per row. CurrentGeneration has rows running from 1 to PopulationSize, where PopulationSize is the number of individuals or chromosomes maintained in each generation. CurrentGeneration has columns running from 0 to NumberOfDecisionVariables, where NumberOfDecisionVariables is the number of variables at play in the model for the GA runs. Column 0 of CurrentGeneration holds the ID of the corresponding chromosome.

Second, the array AbsoluteFitness holds the results of the fitness evaluations for each chromosome in the current generation. AbsoluteFitness has rows running from 1 to PopulationSize and a row of AbsoluteFitness corresponds to a row of CurrentGeneration. AbsoluteFitness has columns from 1 to OutputSize, where OutputSize is the number of distinct values returned for a single chromosome by evaluation of the fitness function. Usually, OutputSize will equal 1, that is, only 1 value is returned: the absolute fitness of the chromosome at hand. Sometimes, however, it is useful to have the fitness function return several values. If so, then their number is indicated by OutputSize and it is the responsibility of the fitness function, Sub Evaluate(I), to organize the response. By convention, column 1 of AbsoluteFitness must hold the absolute (or raw) fitness of the chromosome at hand.

BasicGA works by initializing CurrentGeneration, calculating fitnesses with Evaluate(I) and thereby populating AbsoluteFitness. Then the next generation is created. Crossingover is performed, mutation is performed, and the cycle continues until the stopping condition (a count of the generations in this code) is encountered. All this mostly happens through Sub RunGAUntilDone.

Now some specific comments about these declarations.

1. The following parameter is set in InitializeGA:

(a) CurrentIDNum. An integer, representing the ID number, or count, of a given chromosome or solution.
APPENDIX A. BASICGA: CODE FOR GENETIC ALGORITHMS

2. The following parameters are set in GetGARunPars:
   
   (a) **NumberOfGenerations.** Integer, should be $\geq 0$.
   (b) **PopulationSize.** Integer, should be $\geq 1$.
   (c) **CrossoverRate.** Floating point, should be $\in [0, 1]$.
   (d) **MutationRate.** Floating point, should be $\in [0, 1]$.
   (e) **BestNSaved.** Integer, should be $\geq 0$.

3. The following parameters are set in GetGAModelRunPars:
   
   (a) **NumberOfDecisionVariables.** Integer, should be $\geq 1$. This is the number of input variables sent to the fitness evaluation function.
   (b) **OutputSize.** Integer, should be $\geq 1$. This is the number of output values returned by the fitness evaluation function.

4. The following parameters are set in ReDimGAArrays:
   
   (a) **CurrentGeneration.** Declared here as nonstatic, i.e.,
       Dim CurrentGeneration() As Double.
   (b) **AbsoluteFitness.** Declared here as nonstatic, i.e.,
       Dim AbsoluteFitness() As Double.
   (c) **RelativeFitness.** Declared here as nonstatic, i.e.,
       Dim RelativeFitness() As Double.
   (d) **BestNCurrentSaveSet.** Declared here as nonstatic, i.e.,
       Dim BestNCurrentSaveSet() As Double.
A.3 Sub DoTheGA: Code Structure Overview

Sub DoTheGA is the intended entry point to this program. Its structure is quite simple and the source code is given in Figure A.1.

' ******************* Main Program **********
',
Sub DoTheGA ()

Randomize (17)
ChDir "c:\clasave\"
NoisyOutput = 1

' 1. Make preparations to run the GA.

PrepareGA

' 2. Run the GA until the stopping condition is met

RunGAUntilDone

' 3. Postpare the system

PostpareGA
End Sub

Figure A.1: Sub DoTheGA Source Code: Main Entry Point

A few comments are in order. The purpose of Randomize (17) is to initialize the random number generator. This guarantees that on each run the same sequence of random numbers will be generated, regardless of which machine the program is run on.

ChDir "c:\clasave\"

is for the IBM PC (MS DOS) environment and will need to be changed or commented out on the Macintosh. It assumes that a directory called clasave exists on the C drive. The program writes its output files to this directory.
NoisyOutput is set to 1, turning on various comments during the running of the program. Set it to 0 to turn these off.

Now, briefly, to the three subroutines called in Sub DoTheGA.
A.3. SUB DOTHEGA: CODE STRUCTURE OVERVIEW

A.3.1 PrepareGA

The purpose of this subroutine is to initialize the program and to generate the first generation of chromosomes. The source code for this subroutine is given in Figure A.2.

Sub PrepareGA()

' 1. Initialize the system
InitializeGA

' 2. Validate the input data
ValidateGAINput

' 3. Generate the initial population of chromosomes
MakeGAGenOne

' 4. Calculate the absolute and relative 'fitnesses for each chromosome.
CalculateFitness

' 5. Initialize the save sets
InitializeSaveSets
End Sub

Figure A.2: Sub PrepareGA Source Code
A.3.2 RunGAUntilDone

This is the subroutine that does the main work in the program. Its source code is given in Figure A.3.

Sub RunGAUntilDone()
    Do Until NumberOfGenerationsSoFar ≥ NumberOfGenerations
        If (NoisyOutput = 1) Then
            MainForm.ProgressBar.Text =
            "NumberOfGenerationsSoFar = " & NumberOfGenerationsSoFar
        End If
    ' Now to the main business:
    PerformCrossover
    PerformMutation
    CalculateFitness
    UpdateTheSaveSets
    SortBestNCURRENTSaveSet
    NumberOfGenerationsSoFar = NumberOfGenerationsSoFar + 1
    Loop
        If (NoisyOutput = 1) Then
            MainForm.ProgressBar.Text =
            "NumberOfGenerationsSoFar = " & NumberOfGenerationsSoFar
        End If
    End Sub

Figure A.3: Sub RunGAUntilDone Source Code. Note: Lines artificially broken with my continuation symbol: =>.
A.3.3 PostpareGA

Sub PostpareGA cleans things up once the GA has run its course. The program does two things: writes out CurrentGeneration to a file and writes out BestNCurrentSaveSet (the array holding the best N chromosomes found to this point in the GA run) to a file. The source code is given in Figure A.4.

Sub PostpareGA ()

' Print out final generation.
Print2FileCurGen
' Print out the best finds overall.
Print2FileBestOverall

If (NoisyOutput = 1) Then
'   MainForm.ProgressBox.Text = "All done."
End If
End Sub

Figure A.4: Sub PostpareGA Source Code
A.4 PrepareGA: Detailed Code Structure

A.4.1 InitializeGA

InitializeGA initializes CurrentIDNum to 0, then calls three subroutines. The first, GetGARunPars, is for obtaining information needed to make this run of the GA. The second, GetGAModelRunPars, is for obtaining particular information about the model (fitness function) that is to be applied in this particular run of the GA.

The third, ReDimGAArrays, only has the function of setting the sizes of various dynamic arrays (see declarations section, above).

1. CurrentGeneration(1 To PopulationSize, 0 to NumberOfDecisionVariables) As Double.

2. AbsoluteFitness(1 To PopulationSize, 1 To OutputSize) As Double.

3. RelativeFitness(1 To PopulationSize) As Double.

4. BestNCurrentSaveSet(1 To BestNSaved + PopulationSize, 1 To NumberOfDecisionVariables + 1 + OutputSize) As Double

A.4.1.1 GetGARunPars

The following program variables need to be initialized in this subroutine:

1. NumberOfGenerations.


3. CrossoverRate.

4. MutationRate.

5. BestNSaved.

A.4.1.2 GetGAModelRunPars

The following program variables need to be initialized in this subroutine:

1. NumberOfDecisionVariables.
A.4. PREPAREGA: DETAILED CODE STRUCTURE

2. OutputSize.

In addition the following array must be initialized:

1. DecisionVariableInfo.

Specifically,

```vba
ReDim DecisionVariableInfo(1 to _
    NumberOfDecisionVariables, 1 to 4) As Double
```

should be declared and DecisionVariableInfo initialized.

In DecisionVariableInfo each row corresponds to a decision variable. Column 1 holds the LowValue, column 2 the HighValue for the row’s variable. Column 3 is 0 if the variable is not required to be an integer, and 1 otherwise. Finally, column 4 holds grid search information. (BasicGA does not have any grid search capability, but is designed to be expanded.) A 1 indicates that no grid search is being done on that variable. A number larger than 1 indicates that if a grid search is to be done, then the number represents the number of grid points to be examined for that variable. The array holds floating point numbers, and grid search counts are integers. It is up to the grid search program to make the conversion. By convention, we truncate, e.g., 3.1 goes to 3.

A.4.2 ValidateGAInput

The purpose of this subroutine is to validate the information collected in the InitializeGA subroutine. In the current version of the software, little or nothing is done here. Beware!

A.4.3 MakeGAGenOne

Declare: ReDim CurrentGeneration(1 to PopulationSize, 0 to
    NumberOfDecisionVariables) As Double. Each row holds a chromosome of the current generation. Columns 1 through NumberOfDecisionVariables hold values for the corresponding decision variables. Column 0 holds the ID number of the solution.

This subroutine is very simple. It merely uses DecisionVariableInfo to load up CurrentGeneration, with the aid of a random number generator. Also, each member of the generation (i.e., each row) is given a unique ID.
A.4.4 CalculateFitness

This routine calls Evaluate(I) for each member (row) of CurrentGeneration. Evaluate(I) then calculates the fitness of that row and stores it in AbsoluteFitness. By convention, the first column of AbsoluteFitness is the absolute fitness of the corresponding row or solution. If the fitness function, Evaluate(I), returns more than one value, additional values are stored in the second, third, and so on columns of AbsoluteFitness.

Following this, CalculateRelativeFitness is called, which calculates the relative fitnesses from the absolute fitnesses and stores them in RelativeFitness, a one-dimensional array.

A.4.5 InitializeSaveSets

In the basic program, only one save set is used. BestNCurrCurrentSaveSet stores the best $N$ solutions so far, plus the current generation. In this subroutine, CurrentGeneration and AbsoluteFitness are read into BestNCurrCurrentSaveSet, which is then sorted on absolute fitness in the subroutine SortBestNCurrCurrentSaveSet.

A.5 Sub RunGAAuntilDone: Detailed Code Structure

As is clear from the code for Sub RunGAAuntilDone (Figure A.3 and §A.7) this procedure has five main subroutine calls. We now briefly describe each and refer the reader to the complete code listing in §A.7.

A.5.1 PerformCrossover

This is the most complex of the five subroutines, but the basic idea is simple. Using fitness proportional section, two chromosomes are randomly drawn from CurrentGeneration. If crossover is drawn via a random number, then the two chromosomes are crossed over and the results read into the holding array, ChromosomeCopySpace. If crossover is not drawn, then the two chromosomes are simply copied into ChromosomeCopySpace. This continues until
PopulationSize is reached, at which time ChromosomeCopySpace is copied back into CurrentGeneration.

A.5.2 PerformMutation

In this subroutine, the program loops through the entire array CurrentGeneration. For each entry a random number is drawn to determine whether there shall be a mutation. If there is to be a mutation, a uniform random number is drawn between the declared high and low values for the decision variable in question.

A.5.3 CalculateFitness

This routine calls the sub Evaluate which is a model-specific procedure that calculates the values for a row of the array AbsoluteFitness.

A.5.4 UpdateTheSaveSets

Only one save set is present in the program: BestNCurrentSaveSet. The program reads CurrentGeneration into columns 0 through NumberOfDecisionVariables and AbsoluteFitness into the remaining higher-order columns, all this beginning at line BestNSaved + 1. This has the effect of writing over the worst rows of BestNCurrentSaveSet, leaving the best BestNSaved rows intact. The program then (next sub) sorts BestNCurrentSaveSet on absolute fitness.

A.5.5 SortBestNCurrentSaveSet

The program uses a simple bubble sort on column NumberOfDecisionVariables + 1 of BestNCurrentSaveSet. This column is presumed to hold the absolute fitnesses of the various rows.

A.6 Sub PostpareGA: Detailed Code

Structure

Sub PostpareGA calls two subroutines in order to write to files the current generation and the overall best N (= BestNSaved) chromosomes found during
the run of the GA. Source code for these two subroutines is given in Figures A.5 and A.6.

A.7 Complete Code Listing

There follows the complete listing of the code. Following the declarations section, the procedures, whether subs or functions, are in alphabetical order.

Note: For purposes of fitting the listing on the typeset page, I have occasionally broken lines. When I do this, I use the continuation symbol, $==>, which is not part of Visual Basic.
Sub Print2FileBestOverall()
    Dim I, J As Integer
    Dim FNameBestOverall, FNumBestOverall
    Dim msg

    FNumBestOverall = FreeFile
    FNameBestOverall = "B" & NumberOfGenerationsSoFar & "F" & FNumBestOverall & ".TXT"
    Open FNameBestOverall For Output As FNumBestOverall
    For I = 1 To bestNSaved
        msg = ""
        For J = 0 To NumberOfDecisionVariables + OutputSize
            msg = msg & " " & BestNCurrentSaveSet(I, J)
        Next J
        Print #FNumBestOverall, msg
    Next I
    Close

End Sub

Figure A.5: Sub Print2FileBestOverall: Source Code. Note: My continuation symbol, not in Visual Basic: ==>.
Sub Print2FileCurGen()
Dim I, J As Integer
Dim FNameCG, FNumCG
Dim msg

FNumCG = FreeFile
FNameCG = "C" & NumberofGenerationsSoFar &
=> "G" & FNumCG & ".TXT"
Open FNameCG For Output As FNumCG
For I = 1 To PopulationSize
    msg = ""
    For J = 0 To NumberOfDecisionVariables
        msg = msg & " " & CurrentGeneration(I, J)
    Next J
    For J = 1 To OutputSize
        msg = msg & " " & AbsoluteFitness(I, J)
    Next J
    msg = msg & " " & RelativeFitness(I)
    Print #FNumCG, msg
Next I
Close

End Sub

Figure A.6: Sub Print2FileCurGen Source Code.
sok 951127: This is in file: GACODE.BAS in
the folder clapopo3
Am freezing this for now. Call it the BasicGA program,
version 951127.

Option Explicit

'***********************************************************************
'*************** Below, constants declared that ***
'*************** should be read in. ***************

sok 951126: Note: These program variables,
in a non-stubbed
environment, need to be declared in the declarations
section. They are so declared, but I have commented
out the declarations (see below).

'+++++++++++++++++++++++++++++++++++++
'++++ from GetGARunPars +++++++++++

Const NumberOfGenerations = 20
'GetNumberOfGenerations
Const PopulationSize = 100
'GetPopulationSize
Const CrossoverRate = .77
'GetCrossoverRate
Const MutationRate = .23
'GetMutationRate
Const bestNSaved = 100
'GetBestNSaved
'+++++++++++++++++++++++++++++++++++++
'++++++ from GetModelRunPars +++++++++
Const NumberOfDecisionVariables = 4

'GetOutputSize
Const OutputSize = 2

'++++++++++++++++++++++++++++++++++++++
Dim DecisionVariableInfo(1 To NumberOfDecisionVariables, 1 To 4) As Double

Const DecisionVariableInfo11 = 5 'r, low
Const DecisionVariableInfo12 = 20 'r, high
Const DecisionVariableInfo13 = 0 'r, not integer
Const DecisionVariableInfo14 = 0 'r, no grid search

Const DecisionVariableInfo21 = 10 'v, low
Const DecisionVariableInfo22 = 30 'v, high
Const DecisionVariableInfo23 = 0 'v, not integer
Const DecisionVariableInfo24 = 0 'v, no grid search

Const DecisionVariableInfo31 = 15 'u, low
Const DecisionVariableInfo32 = 25 'u, high
Const DecisionVariableInfo33 = 0 'u, not integer
Const DecisionVariableInfo34 = 0 'u, no grid search

Const DecisionVariableInfo41 = 200 'l, low
Const DecisionVariableInfo42 = 300 'l, high
Const DecisionVariableInfo43 = 0 'l, not integer
Const DecisionVariableInfo44 = 0 'l, no grid search

' Global variables

' ++++ from GetGARunPars
' **** but explicitly declared above +++++++

'Dim NumberOfGenerations as Integer
'Dim PopulationSize as Integer
A.7. COMPLETE CODE LISTING

'Dim CrossoverRate As Double
'Dim MutationRate As Double
'Dim bestNSaved as Integer

' +++++++ from GetModelRunPars
' +++++++but explicitly declared above ++++++++.

'Dim NumberOfDecisionVariables As Integer
'Dim OutputSize As Integer

' ++++++++++++++++++++++++++++++++.

Dim Index As Integer
Global CurrentGeneration() As Double
Global AbsoluteFitness() As Double
Dim ChromosomeCopySpace() As Double
Dim RelativeFitness() As Double
Dim CrossoverLikelihood() As Double
Dim BestNCurrentSaveSet() As Double
Dim LowestAbsoluteFitness As Double
Dim HighestAbsoluteFitness As Double
Dim CurrentIdNum As Double
Dim NumberOfGenerationsSoFar As Integer
Dim CrossoverPoint As Integer

Dim NoisyOutput As Integer ' 1 = show lots of output;
' 0 = don’t
Sub CalculateFitness ()
Dim I As Integer

For I = 1 To PopulationSize
    Evaluate (I)
Next I

CalculateRelativeFitness

End Sub
Sub CalculateRelativeFitness()
    Dim I As Integer
    Dim Interval, LowestAbsoluteFitness,
        => HighestAbsoluteFitness As Double

    LowestAbsoluteFitness = FindLowest()
    HighestAbsoluteFitness = FindHighest()
    Interval = HighestAbsoluteFitness - LowestAbsoluteFitness
    If HighestAbsoluteFitness < LowestAbsoluteFitness Then
        MsgBox "Whoa! In CalculateRelativeFitness,
        => HighestAbsoluteFitness = " & HighestAbsoluteFitness & " and
        => LowestAbsoluteFitness = " & LowestAbsoluteFitness
    End If
    For I = 1 To PopulationSize
        If Interval > .00000001 Then
            RelativeFitness(I) = (AbsoluteFitness(I, 1) -
            => LowestAbsoluteFitness) / Interval
        Else
            RelativeFitness(I) = 1
        End If
    Next I
End Sub

Sub CopyStrings(String1, String2, Index)
    Dim I As Integer

    For I = 0 To NumberOfDecisionVariables
        ChromosomeCopySpace(Index, I) =
        => CurrentGeneration(String1, I)
        ChromosomeCopySpace(Index + 1, I) =
        => CurrentGeneration(String2, I)
    Next I
End Sub

Function Crossover() As Integer
Dim ReturnValue As Integer

If Random01Value() <= CrossoverRate Then
    ReturnValue = 1
Else
    ReturnValue = 0
End If
Crossover = ReturnValue
End Function

Sub CrossoverStrings (String1, String2, Index)
Dim I As Integer

CrossoverPoint = Int(((Random01Value() *
=> (NumberOfDecisionVariables - 1)) + 1)
If CrossoverPoint >= NumberOfDecisionVariables Then
    MsgBox "Whoa! In CrossoverStrings."
End If

' Copy up to the crossover point
For I = 1 To CrossoverPoint
    ChromosomeCopySpace(Index, I) =
    => CurrentGeneration(String1, I)
    ChromosomeCopySpace(Index + 1, I) =
    => CurrentGeneration(String2, I)
Next I

' Copy past the crossover point to the end
For I = CrossoverPoint + 1 To NumberOfDecisionVariables
    ChromosomeCopySpace(Index, I) =
    => CurrentGeneration(String2, I)
    ChromosomeCopySpace(Index + 1, I) =
    => CurrentGeneration(String1, I)
Next I

' Assign new IDs to the chromosomes
ChromosomeCopySpace(Index, 0) = GetCurrentIDNum()
ChromosomeCopySpace(Index + 1, 0) = GetCurrentIDNum()
End Sub

' ******************* Main Program ***********
'
Sub DoTheGA()

Randomize (17)
ChDir "c:\clasave"
NoisyOutput = 1

'1. Make preparations to run the GA.

PrepareGA

' 2. Run the GA until the stopping condition is met

RunGAUntilDone

' 3. Postpare the system

PostpareGA
End Sub

Sub Evaluate(I)

' Note: This is a model-specific routine.
' And should be revised, e.g.
' p1 goes to r
Dim p1, p2, p3, p4 As Double
p1 = CurrentGeneration(I, 1)
p2 = CurrentGeneration(I, 2)
p3 = CurrentGeneration(I, 3)
p4 = CurrentGeneration(I, 4)

AbsoluteFitness(I, 1) = 2 * p1 * (1 + p2 / p3) / p4
AbsoluteFitness(I, 2) = 2 * p1 * (1 + p2 / p3) / p4
End Sub

Function FindHighest () As Double
Dim I As Integer
Dim Highest As Double
Highest = AbsoluteFitness(1, 1)
For I = 1 To PopulationSize
    If AbsoluteFitness(I, 1) > Highest Then
    Highest = AbsoluteFitness(I, 1)
Next I
FindHighest = Highest
End Function

Function FindLowest () As Double
Dim I As Integer
Dim Lowest As Double
    Lowest = AbsoluteFitness(1, 1)
For I = 1 To PopulationSize
    If AbsoluteFitness(I, 1) < Lowest Then
    Lowest = AbsoluteFitness(I, 1)
Next I
FindLowest = Lowest
End Function

Function GetCurrentIDNum () As Double
    CurrentIDNum = CurrentIDNum + 1
    GetCurrentIDNum = CurrentIDNum
End Function

Sub GetGARunPars ()
    ' This is a stub right now, with the program variables to be
    ' initialized here declared as constants in the
    ' declarations section.
    ' But here they are:
    ' Const NumberOfGenerations = 2
    ' GetNumberOfGenerations
'Const PopulationSize = 50
    'GetPopulationSize
'Const CrossoverRate = .77
    'GetCrossoverRate
'Const MutationRate = .23
    'GetMutationRate
'Const BestNSaved = 50
    'GetBestNSaved
    NumberOfGenerationsSoFar = 0

End Sub

Sub GetModelRunPars()
    ' This is a stub right now, with the program variables to be
    ' initialized here declared as constants in the
    ' declarations section.
    ' But here they are:
    
    'NumberOfDecisionVariables = 4
    'GetNumberOfDecisionVariables
    'OutputSize = 2
        'GetOutputSize

StubInitDVarInfo
    'for InitDVarInfo
End Sub

Sub InitializeGA()

CurrentIdNum = 0

GetGARunPars
GetModelRunPars
ReDimGAArrays

End Sub
A.7. COMPLETE CODE LISTING

Sub InitializeSaveSets()
Dim I, J As Integer

' Number of rows is the number in the best N save set
' plus the population size
' Number of columns is no. decision variables + ID +
' absolute fitness

' Read in CurrentGeneration array
For I = 1 To PopulationSize
    For J = 0 To NumberOfDecisionVariables
        BestNCurrentSaveSet(I, J) = CurrentGeneration(I, J)
    Next J
Next I

' Read in AbsoluteFitness array
' Note: In the BestNCurrentSaveSet array
' the absolute fitness is
' kept in column number NumberOfDecisionVariables + 1.
For I = 1 To PopulationSize
    For J = 1 To OutputSize
        BestNCurrentSaveSet(I, NumberOfDecisionVariables +
        ==> J) = AbsoluteFitness(I, J)
    Next J
Next I
SortBestNCurrentSaveSet
End Sub

Sub MakeGAGenOne()
Dim I, J As Integer
Dim LowValue, HighValue As Double

For I = 1 To PopulationSize
    For J = 1 To NumberOfDecisionVariables
        LowValue = DecisionVariableInfo(J, 1)
        HighValue = DecisionVariableInfo(J, 2)
        CurrentGeneration(I, J) =
        ==> RandomBetween(LowValue, HighValue)
    Next J
Next I
CurrentGeneration(I, 0) = GetCurrentIDNum()
Next I

End Sub

Sub PerformCrossover ()
Dim I, J As Integer
Dim String1, String2 As Integer
Dim SumFitneses As Double

SumFitneses = 0
For I = 1 To PopulationSize
    SumFitneses = SumFitneses + RelativeFitness(I)
Next I
' CrossoverLikelihood accumulates the probabilities of
' crossover. So,
' CrossoverLikelihood(PopulationSize) should = 1.
CrossoverLikelihood(I) = RelativeFitness(I) / SumFitneses
For I = 2 To PopulationSize
    CrossoverLikelihood(I) =
    ==> (RelativeFitness(I) / SumFitneses) +
    ==> CrossoverLikelihood(I - 1)
Next I
For I = 1 To PopulationSize Step 2
    String1 = RandomStrings()
    ' get a random string that can be crossed over
    String2 = RandomStrings()
    ' get a random string that can be crossed over
    If Crossover() = 1 Then
        ' If we do crossover here, then
        CrossoverStrings String1, String2, I
    Else ' We don't do crossover and
        ' we just copy the chromosomes to the next generation.
        Call CopyStrings(String1, String2, I)
    End If
Next I

' copy back into the CurrentGeneration array
For I = 1 To PopulationSize
    For J = 0 To NumberOfDecisionVariables
        CurrentGeneration(I, J) =
          => ChromosomeCopySpace(I, J)
    Next J
Next I
End Sub

Sub PerformMutation()
    Dim I, J As Integer

    For I = 1 To PopulationSize
        For J = 1 To NumberOfDecisionVariables
            If Random01Value() < MutationRate Then
                CurrentGeneration(I, J) =
                  => RandomBetween(DecisionVariableInfo(J, 1),
                  => DecisionVariableInfo(J, 2))
                      CurrentGeneration(I, 0) = GetCurrentIDNum()
            End If
        Next J
    Next I
End Sub

Sub PostpareGA()

    ' Print out final generation.
    Print2FileCurGen
    ' Print out the best finds overall.
    Print2FileBestOverall

    If (NoisyOutput = 1) Then
        MainForm.ProgressBox.Text = "All done."
    End If

End Sub

Sub PrepareGA()
1. Initialize the system

    InitializeGA

2. Validate the input data

    ValidateGAINput

3. Generate the initial population of chromosomes

    MakeGAGenOne

4. Calculate the absolute and relative fitnesses for each chromosome.

    CalculateFitness

5. Initialize the save sets

    InitializeSaveSets

End Sub

Sub Print2FileBestOverall ()
Dim I, J As Integer
Dim FNameBestOverall, FNumBestOverall
Dim msg

FNumBestOverall = FreeFile
FNameBestOverall = "B" & NumberOfGenerationsSoFar &
    "F" & FNumBestOverall & ".TXT"
Open FNameBestOverall For Output As FNumBestOverall
For I = 1 To bestNSaved
    msg = ""
    For J = 0 To NumberOfDecisionVariables + OutputSize
        msg = msg & " " & BestNCurrentSaveSet(I, J)
Next J
    Print #NumBestOverall, msg
Next I
Close

End Sub

Sub Print2FileCurGen()
    Dim I, J As Integer
    Dim FNameCG, FNumCG
    Dim msg

    FNumCG = FreeFile
    FNameCG = "C" & NumberOfGenerationsSoFar &
    ==> "G" & FNumCG & ".TXT"
    Open FNameCG For Output As FNumCG
    For I = 1 To PopulationSize
        msg = ""
        For J = 0 To NumberOfDecisionVariables
            msg = msg & " " & CurrentGeneration(I, J)
        Next J
        For J = 1 To OutputSize
            msg = msg & " " & AbsoluteFitness(I, J)
        Next J
        msg = msg & " " & RelativeFitness(I)
        Print #FNumCG, msg
    Next I
Close

End Sub

Function Random01Value()
    ' Note: Here and only here we use the 0-1
    ' random number generator built into Basic.

    Random01Value = Rnd
    ' return a random value from the interval [0,1]
End Function

Function RandomBetween (Low, High)
    RandomBetween = (Random01Value() * (High - Low)) + Low
End Function

Function RandomStrings ()
    ' The purpose of this routine is to pick
    ' a chromosome to contribute to the next
    ' generation. The likelihood of being picked
    ' is proportional to the relative fitness of the
    ' chromosome
    Dim I As Integer
    Dim PointOnUnitInterval As Double

    PointOnUnitInterval = Random01Value()
    I = 1
    While CrossoverLikelihood(I) < PointOnUnitInterval
        I = I + 1
    Wend

    RandomStrings = I
End Function

Sub ReDimGAArrays ()
    ReDim CurrentGeneration(1 To PopulationSize,
        ==> 0 To NumberOfDecisionVariables) As Double
    ReDim ChromosomeCopySpace(1 To PopulationSize,
        ==> 0 To NumberOfDecisionVariables) As Double
    ReDim AbsoluteFitness(1 To PopulationSize,
        ==> 1 To OutputSize) As Double
    ReDim RelativeFitness(1 To PopulationSize) As Double
    ReDim BestNCurrentSaveSet(1 To bestNSaved +
        ==> PopulationSize, 0 To NumberOfDecisionVariables +
A.7. COMPLETE CODE LISTING

```vba

Sub RunGAUntilDone ()
    Do Until NumberOfGenerationsSoFar >= NumberOfGenerations
        If (NoisyOutput = 1) Then
            MainForm.ProgressBar.Text = 
            => "NumberOfGenerationsSoFar = "
            => & NumberOfGenerationsSoFar
        End If
        ' Now to the main business:
        PerformCrossover
        PerformMutation
        CalculateFitness
        UpdateTheSaveSets
        SortBestNCurrentSaveSet
        NumberOfGenerationsSoFar = NumberOfGenerationsSoFar + 1
    Loop
    If (NoisyOutput = 1) Then
        MainForm.ProgressBar.Text = 
        => "NumberOfGenerationsSoFar = "
        => & NumberOfGenerationsSoFar
    End If
End Sub

Sub SortBestNCurrentSaveSet ()

Dim CurrentRow, I As Integer
Dim ArraySize As Integer
Dim SortIndex As Integer
Dim NumberSwapped As Long

NumberSwapped = -1
ArraySize = bestNSaved + PopulationSize
```
SortIndex = NumberOfDecisionVariables + 1
   ' Above: note that in InitializeSAveSets that
   ' the absolute fitness is read
   ' into column NumberOfDecisionVariables + 1
While NumberSwapped <> 0
   NumberSwapped = 0

   For CurrentRow = 1 To ArraySize
      I = CurrentRow
      While I <= ArraySize
         If BestNCurrentSaveSet(CurrentRow, SortIndex) <
            => BestNCurrentSaveSet(I, SortIndex) Then
               SwapRows CurrentRow, I
               NumberSwapped = NumberSwapped + 1
            End If
         I = I + 1
      Wend
      Next CurrentRow
   Wend

End Sub

Sub StubInitDVarInfo ()
Dim I, J As Integer

   ' Load up the array DecisionVariableInfo

   'For I = 1 To NumberOfDecisionVariables
      '   For J = 1 To 4

         '   Next J
      'Next I

   DecisionVariableInfo(1, 1) = DecisionVariableInfo1
      'r, low
   DecisionVariableInfo(1, 2) = DecisionVariableInfo12
      'r, high
   DecisionVariableInfo(1, 3) = DecisionVariableInfo13
A.7. COMPLETE CODE LISTING

' r, not integer
DecisionVariableInfo(1, 4) = DecisionVariableInfo14
' r, no grid search

DecisionVariableInfo(2, 1) = DecisionVariableInfo21
' v, low
DecisionVariableInfo(2, 2) = DecisionVariableInfo22
' v, high
DecisionVariableInfo(2, 3) = DecisionVariableInfo23
' v, not integer
DecisionVariableInfo(2, 4) = DecisionVariableInfo24
' v, no grid search

DecisionVariableInfo(3, 1) = DecisionVariableInfo31
' u, low
DecisionVariableInfo(3, 2) = DecisionVariableInfo32
' u, high
DecisionVariableInfo(3, 3) = DecisionVariableInfo33
' u, not integer
DecisionVariableInfo(3, 4) = DecisionVariableInfo34
' u, no grid search

DecisionVariableInfo(4, 1) = DecisionVariableInfo41
' l, low
DecisionVariableInfo(4, 2) = DecisionVariableInfo42
' l, high
DecisionVariableInfo(4, 3) = DecisionVariableInfo43
' l, not integer
DecisionVariableInfo(4, 4) = DecisionVariableInfo44
' l, no grid search

End Sub

Sub SwapRows (Index1, Index2)

Dim I As Integer
Dim Temp() As Double
ReDim Temp(0 To NumberOfDecisionVariables +


APPENDIX A. BASICGA: CODE FOR GENETIC ALGORITHMS

```vbnet
>> OutputSize) As Double

    For I = 0 To NumberOfDecisionVariables + OutputSize
        Temp(I) = BestNCurrentSaveSet(Index1, I)
    Next I

    For I = 0 To NumberOfDecisionVariables + OutputSize
        BestNCurrentSaveSet(Index1, I) =
>> BestNCurrentSaveSet(Index2, I)
    Next I

    For I = 0 To NumberOfDecisionVariables + OutputSize
        BestNCurrentSaveSet(Index2, I) = Temp(I)
    Next I

End Sub

Sub UpdateTheSaveSets()
    Dim I, J As Integer

    ' Basically, this subroutine dumps the
    ' CurrentGeneration array
    ' and the AbsoluteFitness array into the
    ' BestNCurrentSaveSet array, by appending them after the
    ' current bestNSaved. Later, we sort the entire array.
    ' This is done as the next subroutine call in RunGAUntilDone.

    ' Right now only the best N overall save set
    ' Number of rows is the number in the save set plus
    ' the population size
    ' Number of columns is no. decision variables +
    ' ID + absolute fitness
    For I = 1 + bestNSaved To PopulationSize + bestNSaved
        For J = 0 To NumberOfDecisionVariables
            BestNCurrentSaveSet(I, J) =
>> CurrentGeneration(I - bestNSaved, J)
        Next J
        For J = 1 To OutputSize
```
A.8  FILE NOTES

BestNCurrentSaveSet(I,
===> NumberOfDecisionVariables + J) =
===> AbsoluteFitness(I - bestNSaved, J)
    Next J
Next I
End Sub

Sub ValidateGAInput ()
' Note: There’s a lot more that needs doing here.

    If NumberOfGenerationsSoFar > NumberOfGenerations Then
        MsgBox "NumberOfGenerationsSoFar > NumberOfGenerations " &
    ==> NumberOfGenerationsSoFar & " " & NumberOfGenerations
    End If

End Sub

A.8  File Notes

April 8, 1998: Brought this file, dt-basicga.tex (formerly latex), over from the
Macintosh. It’s history there: [File: dt-basicga.latex. Created: November
27, 1995, from clam1-code.latex. Revised: 951222, 951128.]