GENETIC ALGORITHM LEARNING MODULE
FOR ENVIRONMENTAL ENGINEERING SYSTEMS

by
Erin M. Satchell

A REPORT
Submitted in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE IN ENVIRONMENTAL ENGINEERING

MICHIGAN TECHNOLOGICAL UNIVERSITY
2009
This report, “Genetic Algorithm Learning Module for Environmental Engineering Systems”, is hereby approved in partial fulfillment of the requirements for the Degree of Master of Science in ENVIRONMENTAL ENGINEERING.

Department:

Civil and Environmental Engineering

Signatures:

Report Advisor: ________________________________
David Watkins

Department Chair: ________________________________
William Bulleit

Date: ________________________________
Abstract

As engineering students are introduced to methods for systems analysis, they often proceed through a series of techniques – such as linear programming and the numerical approximation of derivatives – which can be comparatively straightforward for the instructor to illustrate and the student to implement. Addressing evolutionary computing techniques in an introductory course can be more challenging, since they require greater computer programming ability. This learning module is designed to assist the instructor in providing students with hands-on exposure to evolutionary computing. A genetic algorithm is presented with simplified examples, accessible annotated code, and supplementary reading material. A demonstration application of the algorithm to a groundwater management problem is also provided.
# Table of Contents

Abstract ........................................................................................................................................ iii
Introduction .................................................................................................................................... 1
Learning Module Overview ........................................................................................................... 3
   Software ..................................................................................................................................... 3
   User’s Manual ............................................................................................................................ 3
   Demonstration Files .................................................................................................................... 4
   The Genetic Algorithm Program ................................................................................................. 5
Exercises .......................................................................................................................................... 5
Summary .......................................................................................................................................... 6
Appendix A: User’s Manual ........................................................................................................... 8
Appendix B: “Genetic Algorithm Example” Code ........................................................................... 31
Appendix C: “Finite Difference Watershed Example” Code ............................................................ 32
Appendix D: “Watershed Spreadsheet” .......................................................................................... 33
Appendix E: “Genetic Algorithm Watershed Program” Code ......................................................... 35
Appendix F: “Genetic Algorithm Watershed Program” Sample Solution ..................................... 36
Introduction

As students are educated in engineering problem-solving, they are gradually exposed to increasingly complex methods for obtaining solutions. An area of study like water treatment may begin with the introduction of basic concepts and idealized forms such as mass balance and the completely mixed reactor, along with relevant analytical solution methods (Crittenden et al., 2005, p. 360). Progression from the idealized toward more realistic representations typically requires the introduction of more complicated forms and more flexible solution methods. The analytical solution to a differential equation may give way to Euler's method of numerical approximation, for example.

In a similar fashion, the study of operations research or systems analysis may follow a progression in model formulation from linear programming to non-linear programming to integer programming. Specific solution methods are introduced to accompany the various models, such as the simplex method, Lagrange multipliers, and branch-and-bound. The goal of operations research is typically to find an optimal solution, perhaps the best allocation of resources among a set of competing uses (Jensen and Bard, 2003).

As students encounter more recalcitrant problems, such as mathematical programs which are both non-linear and discontinuous, a class of solution methods may be introduced which relinquish the deterministic search for global optima and, instead, attempt the efficient exploration of the feasible solution spaces. Although such search methods do not guarantee the discovery of a best possible solution, they represent a pragmatic compromise which can deliver good solutions to problems for which the extraction of a definitive optimum is impractical. Among these search methods is an evolutionary computing approach called genetic algorithms (Eiben and Smith, 2003).
My undergraduate experience with genetic algorithms provided the impetus for creating this learning module. A short introduction to the method opened my eyes to a way of approaching problems that was not reliant on mathematical structures built up from comprehensive foundations of first principles. The new method appeared much simpler to implement for many cases. The basic algorithm could be adapted for many dissimilar problems. And small problems which could not justify a large programming effort might be addressed.

I wanted to further explore genetic algorithms, but found a gap in the available learning materials. Between the basic overviews in introductory texts and the full-blown applications found in advanced texts, there seemed to be little bridging material targeted at the novice. This learning module is intended to assist in filling that gap by providing additional materials for instructors in survey courses on systems modeling, numerical methods, or operations research.

The learning module gives students a hands-on experience with a genetic algorithm application, without requiring that they have computer programming experience. Graphical program output promotes understanding by illustrating key concepts (Jacob, 2001) and providing visual feedback as students explore. Augmenting traditional lecture-based classes with activities in which the student may take a more active and exploratory role can improve both student satisfaction and the learning outcome (Barrosso and Morgan, 2009).

Genetic algorithms are appropriate for addressing a wide range of problems in civil and environmental engineering, such as Bau and Mayer, 2008. They are especially useful for problems not readily expressed in linear terms, when a feasible region is non-convex, or where there are other complicating factors. Though flexible, there are costs to using the algorithms, however. Processing times can be long and global optimality is not ensured. But with rising computing speeds, genetic algorithms become increasingly practical methods for finding good solutions to intransigent problems.
Learning Module Overview

Software
The learning module utilizes Mathematica® software from Wolfram Research, Inc (Wolfram, 2009). The Mathematica environment facilitates a user-friendly application which does not require any knowledge of the programming language, while still providing immediate access to the underlying code and intermixed textual notes for the interested user. As of December 2009, it is also one of the few software titles which is available campus-wide in its full-capability version for free download to student and faculty home computers, as well as to on-campus computers. Also included in the learning module are files created with Microsoft, Inc.’s Excel® and Word® software.

User’s Manual
The User’s Manual contained in Appendix A is intended as a stand-alone guide for the student. Too little information can leave the student floundering or “stuck” on programming idiosyncrasies that do little to advance their understanding. Too much information can be overwhelming, requiring the student to assimilate a large volume of facts before grasping the essential outlines of a topic. The manual attempts to find a productive balance by presenting information to students in stages.

The manual first gives the student an overview of a topic. The student is then guided to the computer program relating to that topic. As the student explores the program, they are presented with a series of structured commentaries – each held beneath unobtrusive headings until the student elects to view them – which explain the workings of a section of the code and how that section relates to the function of the overall algorithm. This arrangement allows the student to focus their attention on individual aspects of the algorithm in an orderly way, without the distraction of navigating large blocks of code.
**Demonstration Files**

When first encountering a new concept, it can be helpful to see that concept applied in a familiar setting. For this reason, the learning module begins with the “Genetic Algorithm Example”. This example is an application of a genetic algorithm to the solution of a simple problem with a single optimal solution which is readily apparent to the user. Knowing in advance the best answer to the question, the student is invited to explore and manipulate parameters of the genetic algorithm. The “Genetic Algorithm Example” program accepts the student’s parameter values and records its progress toward a solution. The archived processing record is then rendered graphically for the student, to make the internal workings of the program and the effects of the student’s parameter adjustments more readily apparent.

Following the “Genetic Algorithm Example”, the student is invited to explore the “Finite Difference Watershed Example”, which introduces the student to a steady-state groundwater elevation map for an idealized watershed. Various arrangements of wells and pumping rates within the watershed may be entered by the student. The example program then uses a finite difference approach to solve for the steady-state groundwater elevations and display those elevations as a three-dimensional graphic.

The “Watershed Spreadsheet” is intended facilitate the student’s own manual search for a solution to the same watershed problem that the “Genetic Algorithm Watershed Program” addresses. Using the spreadsheet, the student can propose a solution to the problem and then manually enter that solution into the “Finite Difference Watershed Example” to see the resulting groundwater profile. With these tools, the student is able to efficiently explore the problem for themselves, before moving on to a computer-generated solution from the “Genetic Algorithm Watershed Program”.
The Genetic Algorithm Program

By the time the student arrives at the “Genetic Algorithm Watershed Program”, they will be familiarized with the use of Mathematica, the workings of a genetic algorithm, and the specific watershed problem that is being addressed. The program code is taken directly from the “Genetic Algorithm Example” and the “Finite Difference Watershed Example”, with any necessary modifications noted. The program works on the same problem that the students undertook manually, allowing the students to make comparisons between the efficiency of the computer’s evolutionary search and their own directed search patterns.

Exercises

At the conclusion of each section of the User’s Manual, a number of exercises are presented. These exercises range from straightforward tasks, with guidance for the student, to more challenging open-ended questions.

Upon initially opening the learning module programs, the program parameters are deliberately set to values that are not very efficient. If few exercises are to be assigned, the instructor may wish to emphasize those that would have the student discover more efficient model parameters. If those exercises are not to be assigned, the instructor might consider changing the initial parameter settings within the code before distributing it to students or providing the students with a list of more efficient settings.
Summary

Genetic algorithms are potentially very adaptable and powerful tools. But they can be difficult to demonstrate in a classroom setting, particularly for students who do not possess a background in computer programming. This genetic algorithm learning module provides instructors with tools that will allow students to learn about genetic algorithms through hands-on interaction as they find solutions to an environmental engineering problem.

The learning module is designed to progressively introduce students to key concepts. Ideas are presented in stages through the user’s manual, commentary contained in the software, and exercises intended to guide the students in considering how a genetic algorithm functions and in identifying its strengths and weaknesses. The code itself is extensively annotated, and will hopefully be found accessible to those wanting to inspect the workings of these learning module programs or to adapt them for their own uses.
References


Appendix A: User’s Manual
User’s Manual

to accompany

“Genetic Algorithm Learning Module for Environmental Engineering Systems”

December, 2009
Getting Started

It is hoped that the reader will have fun. Nothing takes the “thrill” out of new software faster than getting stuck with no help in sight. This manual attempts to guide the reader in an ordered way through each procedure. So, to quote Douglas Adam’s exemplary guidebook, “Don’t panic!”

The first program you will open is titled “Genetic Algorithm Example”. It is written in Wolfram Research, Inc.’s Mathematica® computing environment. As of December 2009, Mathematica software is licensed for free download by all MTU students to their personal computers, and is also widely available on campus computers. The Mathematics Department provides a link to the download site at: http://www.mathlab.mtu.edu/mediawiki/index.php/Main_Page

Once there, you will need to scroll down the page to find the correct link. Clicking on the link, you will be prompted for your MTU username and password. You may also receive a warning that the site’s security certificate is invalid. I have ignored that warning for years at this site, with no visible ill effects. Once you have logged in, you will be taken to a page listing 3 steps for downloading. You will also be confronted with a list of 20 different download options. If you are unsure which to choose, the “.zip” option for the most recent version in Windows is a likely choice.

If you are using a computer in one of the labs instead of downloading to your own machine, you may have trouble locating Mathematica. If you know it is there somewhere, you might just try opening one of the “.nb” files and allow the computer to locate Mathematica via the “notebook” file type. It has been my experience, however, that installation in the labs is sometimes spotty and varies by department, by lab, and by which seat you are in at the moment. The lab staff can always add Mathematica to a computer or find one for you that already has it installed.
About Genetic Algorithms

A genetic algorithm is a method for solving problems. A random search might be the first problem-solving process one discovers as a child, but the budding student is soon introduced to more efficient problem-tailored methods. The student learns to classify a problem and to draw the appropriate solution method from their expanding repertoire (sometimes regressing to random search on multiple-choice exams). Analytical tools progress from arithmetic to algebra and then calculus, and are eventually enhanced by numerical approximation techniques and systems for manipulating large sets of equations.

Though effective, there are costs to using the problem-tailored methods. Non-linearity and discontinuity can be very troublesome. And a lot of problem-specific effort may be required in framing each individual case. A problem-tailored solution is akin to a custom-fitted, hand-stitched suit of clothes – requiring greater effort and skill, but delivering the finest results. The genetic algorithm solution is closer to machine-made, one-size-fits-all clothing. The fit’s not as good, but with a drawstring and some elastic a serviceable garment can be had at a low price. Figure 1, below, conceptually compares performance and range of coverage for a random search, a problem-tailored method, and an evolutionary algorithm.

Figure 1: Evolutionary algorithm performance
(Figure adapted from Introduction to Evolutionary Computing, A.E. Eiben and J.E. Smith, 2003, p. 32)
Evolutionary algorithms help find solutions to problems using search methods inspired by processes observed in the natural world. A genetic algorithm is one type of evolutionary algorithm. With a genetic algorithm, individual solutions exist in an environment that is defined by the constraints and goals of a specific problem, much as natural organisms exist in an environment defined by the “laws of nature”. The attributes of each solution are coded in that “individual's” genes.

Pairs of individuals within a population of solutions can combine, and confer new combinations of their genetic attributes upon offspring. And mutation can carry those attributes in new directions. As with natural selection, the attributes of each individual determine whether they are comparatively more or less “fit” within their particular environment. The “more fit” individuals are more successful at genetically influencing the population of solutions through their offspring, allowing the population to evolve over time. As with evolution in the natural world, it is not necessary to fully understand an environment for the process to proceed. Unexpected, novel solutions may arise. And there is a strong element of chance.

**Starting Out with the Genetic Algorithm Example**

Once you have secured a computer with Mathematica installed, you should open the “Genetic Algorithm Example” notebook file. Depending on how your version of Mathematica is configured, one of two things will happen. If a message box pops up asking whether you would like to “automatically evaluate all the initialization cells”, you should click “Yes”. This will direct Mathematica to evaluate the code and launch this example's control console.

If a message box does not pop up of its own accord, you will need to prod things along a little. There are a few ways to initiate the evaluation manually. Clicking on the “Evaluation” tab on the main Mathematica toolbar will present you with a list of options. Select “Evaluate Notebook”.

12
Another way to tell *Mathematica* to evaluate something is to click on the cell bracket you wish to evaluate and then press “Shift-Enter”. This is especially handy if you want to run only selected parts of the code. The cell brackets are found at the extreme right of the notebook window. To evaluate the entire notebook using this method, click on the outermost group bracket that extends across all the other brackets and press “Shift-Enter”.

The cell brackets can tell you a few things. They indicate how the code is organized into sections that *Mathematica* calls “cells”. Some cells hold headings, some text, others code. And groups of cells can be “collapsed” or hidden beneath the heading cells. This allows large amounts of code to be neatly tucked away, making navigation through a program much easier. Whenever you see a cell group bracket that has a “half-arrow” pointing downward, you know that there are other cells hidden beneath that one. Double clicking on the “half-arrow” bracket will show you what is underneath. Double-clicking again closes it back up. In this notebook, clicking on the triangle buttons to the left of the headings will also show you what is beneath a heading.

**Running the Genetic Algorithm Example**

Upon evaluating the notebook, you will be presented with a green control console at the bottom of the screen. It has three buttons: “Run GA”, “Clear Output”, and “Set Parameters”. Human nature being what it is, you probably dearly want to push the “Run GA” button if you have not done so already. But the outcome will make more sense if we first take a look at what is under the headings.

Click on the arrow next to “A Simple Example Problem” and read the problem statement underneath. When you have finished, click the arrow again to close that section back up. As you read, you will notice two kinds of text. The first section relates general information. The second kind of text is contained within “(* *)” symbols and provides commentary on the code itself. The second kind of
text may be skipped over along with the code itself, unless this is of interest to the reader.

Continue reading through each of the sections, except for the final two: “Evaluation Functions” and “Output”. Those sections are optional. They contain code and related commentary, but no expository text.

After reading the sections, some diagrams may help with visualizing how solutions are represented and evolved within this genetic algorithm. The set of values defining each individual solution is represented as a genetic code. For this problem there are two chromosomes: one for investing in stocks and one for investing in bonds. Within those chromosomes are two genes: one indicating whether a brokerage fee has been paid and one indicating how much has been invested.

**Figure 2: Representation**

Two Chromosomes:

- **Stocks**
  - Fee: 0 or 1
  - Amt. Invested: 0 to 100

- **Bonds**
  - Fee: 0 or 1
  - Amt. Invested: 0 to 100

Two Genes:

- **Brokerage Fee**
  (Binary variable indicates Yes or No decision.)

- **Amount Invested**
  (Real variable gives dollar value.)
Mating occurs when two individuals are selected as “parents” from the population. They are labeled “Ma” and “Pa” in the diagram below. These parents create offspring which contain some combination of the genetic information possessed by the parents. When offspring are created, a recombination of the parents genetic information can occur. In the diagram, “Bro” inherited “Ma’s” genetic code with no changes. “Sis”, on the other hand, got her first chromosome unchanged from “Ma” but her second chromosome was from “Pa”. This happened because a crossover occurred that gave “Sis” a combination of the parent genomes. In addition, a mutation occurred in the second gene of the second chromosome, changing the dollar amount invested in bonds to $31.

**Figure 3: Mating, Recombination, and Mutation**

Once a new generation has been created, all of the individuals – both parents and offspring – are combined into a single group. The fitness of each solution is determined, with fitness based upon how well that solution performs in the
problem environment. The fittest solutions survive to become the parents for the next generation.

And now it is now time to push the buttons. Click on “Set Parameters”. This button will bring up a block of user-definable variables. The first column gives their current values and the second column provides spaces where the current values can be changed. These values are being held dynamically in the computer memory, so your changes will take effect immediately and subsequent re-evaluation of the code is not required.

Population Size sets the number of individuals in the parent population. It must be an even number. Number of Generations determines how many times the mating→ offspring→ selection cycle will occur. Mutation Rate is a probability between 0 and 1.0, giving the likelihood that a mutation will occur at any single gene. Mutation Distance influences the size of a mutation, if one were to occur. A larger value indicates a larger maximum potential mutation. Crossover Rate is a probability between 0 and 1.0, giving the likelihood that a crossover will occur at any single gene. The Merdle Penalty is applied to individuals in the population who spend more on either stocks or bonds than the total amount of available cash. The Madoff Penalty is applied to individuals who spend more on stocks and bonds together than the total amount of available cash.

Clicking the “Clear Output” button will clear away output you no longer want. You can also select individual cell brackets and “Delete” them, but just punching the button is faster. You may elect to save some output as a record of your trials. If so the output will continue to extend down the page.

That only leaves only one remaining button to explore. Only the most saintly reader will not have pushed it by now. Click the “Run GA” button and a progress indicator will appear. The indicator is eventually replaced by a series of outputs.
The Best Solution shows the information contained in the genes of the fittest member of the last generation. Fitness gives the year-end dollar value of the best solution. Investment in Stock, Investment in Bonds, and Investment in Cash shows how the initial cash available was apportioned among the three categories.

Below the printed output is a graph with Median Fitness on the vertical axis and Generation Number on the horizontal. The median is the “middle” member of the population. By tracking the evolution of the population’s midpoint, one can see how the population as a whole is developing.

Each time the “Run GA” button is pushed, a new randomly-generated population evolves according to the parameters the user has selected. For this simple problem, that evolution is displayed as a “movie”. In addition to “Play” and other buttons, the user may drag the slider to view selected parts of the “movie”.

The final output item is the total computer processor time used to run the “Genetic Algorithm Example”. Noting this time allows the user to gauge how changes to the parameters have influenced the efficiency of the algorithm.
Exercises for the “Genetic Algorithm Example”

(Notes: The initial population is randomly generated, making every run unique. You may want to run each set-up a few times to get the best sense of what is occurring. Individuals are represented in the movie with blue dots. Dots stacked on top of each other will look like one dot.)

1. Set the crossover rate and the mutation rate to 0 and run the program. View the movie using the slider. What happens? Why?

2. With the mutation at 0, set crossover to 0.5. What happens? Why?

3. With crossover at 0, set mutation to 0.5. What happens? Why?

4. With crossover and mutation at 0.5, set the Madoff Penalty to 0. What happens? Why?

5. On your own, manipulate the parameters so that the algorithm will solve this problem more efficiently. Do you observe any trade-offs between the speed at which the population converges toward a solution and the precision of that solution? Which parameters influence speed? Which influence precision? Why?
Running the “Finite Difference Watershed Example” and the “Watershed Spreadsheet”

Open the “Finite Difference Watershed Example” and evaluate the notebook. (Use “Evaluate Notebook” or “Shift-Enter” as in the previous section.) A computer-generated graphic will appear. It is a representation of groundwater elevations found within a straight-sided watershed, with the vertical dimension exaggerated for clarity. A river flows through the watershed, with the groundwater intersecting the river along the blue lines. The green cylinders represent wells which extend upward to the ground surface. The sizes of the drawdown curves around each well depend on the amount of water being pumped from the wells, the transmissivity of the aquifer, and on the groundwater recharge. The graphic is three-dimensional and can be manipulated as described in the text above the image.

As with the previous “Genetic Algorithm Example”, the commentary and code in this file is arranged beneath section headings. It is not essential to read through all of these sections, unless you are interested in how the finite element model works.

Open the section titled “Creating a pumping Scenario…” In that section you will find these two lines of code:

```plaintext
wells={321,503,877,1441,1466};
pumpingRates={-200,-200,-50,-100,-100};
```

Changing the numbers in the “wells” vector will change the locations of the wells in the watershed. Changing the numbers in the “pumpingRates” vector will change the pumping rates for the wells. The first pumping rate goes with the first well in the “wells” vector, and so on. The negative signs indicate that water is being pumped out of the watershed. (If you leave off the sign, water will be injected into the well. It’s OK if you want to try it.)
Open the section titled “Groundwater Elevations at Well Sites”. You will see this line of code:

\[ \text{MatrixForm}\left[\{x_{[321]}, x_{[503]}, x_{[877]}, x_{[1441]}, x_{[1466]}\}\right] \]

The subscripts within the double brackets ([[321]], etc) indicate the locations of wells. Once the finite difference model has determined the groundwater elevations resulting from a particular well scenario, evaluating this command displays those elevations as:

\[
\begin{pmatrix}
79.5948 \\
80.0632 \\
80.6573 \\
80.2933 \\
80.5416
\end{pmatrix}
\]

The topmost number corresponds to the first well listed in the “wells” vector, and so on. So for this example, the first well is at location 321; it is pumping -200 gallons per minute; and the resulting groundwater elevation at location 321 is 79.59 ft.

There is one last bit of information you need to locate in this file. Open the section titled “Any Constraint Violations?” In answer to the question “Violations?”, you will find either “True” or “False”. There are some environmental rules built into this finite difference model, and this is the place to check whether they have been violated. Pumping too much water, putting a well too close to the river (or in the river) will probably put “True” here after the scenario has been evaluated.

With the “Finite Difference Watershed Example” still open, open up the Excel file titled “Watershed Spreadsheet”. The watershed spreadsheet will allow you to
manually explore the watershed problem that the genetic algorithm will eventually be working on. There are two worksheets. The “Map” worksheet shows the geometry of the watershed. Around the perimeter are yellow cells. These represent the watershed boundary. In the model, no water crosses the boundary. In the far upper-left corner is an orange cell that indicates the location of a town. The green cells correspond to the green wells you already saw in the finite difference model’s example scenario. The light blue cells are the river cells within the watershed. The dark blue cell is the river outlet. The river is assumed to flow into a lake and the groundwater elevation at this point is constant.

Surrounding the watershed map are X and Y coordinate numbers. On the cells within the map, individual grid cell location numbers can be found. The X-Y coordinate system is used in the genetic algorithm so that mutations to a well’s X or Y coordinate can be used to move the well into nearby cells. The grid system is used by the finite difference model. The model creates a system of simultaneous equations, with equations for each square in the grid. A sequential numbering system is needed for referencing the equations in a coefficient matrix. The map will hopefully provide the user with a handy visual cross-reference between the two number systems.

The other worksheet in the “Watershed Spreadsheet” is titled “Cost Calculator”. The values in the “Cost Calculator” are initially those for the finite difference example scenario. The “Cost Calculator” will assist you in your search for the least-cost arrangement for a well field that is to be built in the watershed. Here is a description of the problem:

A small town in a quiet Upper Peninsula watershed wants to develop a well field in order to secure a more reliable source of drinking water. From 1 to 5 wells may be drilled, and there can be only one well at a single site. The maximum size of an individual well is 275 gallons per minute. Each well can have a different pumping capacity, provided that the total capacity of all of the wells will provide a minimum water supply of 850 gpm. The cost to establish each well is $70,000 regardless of well location.
The lifetime operating cost for each well depends upon three factors: the distance from the well to the town, the volume of water pumped from the well, and the amount of vertical lift required to bring the water to the surface.

The town's part-time engineer/postmaster has determined that the net present value of the expected lifetime operating cost for each individual well can be found using a simple non-linear function:

\[ \text{NPV[operating cost]} = (\text{Distance in chains})^{0.4} \times (\text{Pumping Rate in gpm}) \times (\text{Steady state depth to groundwater in feet})^{2.3} \times (\text{An omnibus cost adjustment factor in dollars per chain-gallon-foot}^{2.3} \text{ per minute}) \]

Five chains equals 1/16 of a mile, which is the length of each side of the 2.5-acre squares found on the watershed map. The postmaster measures in chains and acres because he was a farmer prior to studying engineering. The steady-state depth to water is determined using the Finite Difference Watershed Model. The town sits at an elevation of 100 ft. The omnibus cost adjustment factor is $0.01.

The decisions the planners need to make are: How many wells? Where should they be sited? How much water should each well provide?

The objective is to minimize the total lifetime cost of the system.

There are three constraints. One is a total water yield constraint. The planned new water system will only be built if it is projected to provide the required minimum water supply. Keeping the old system will lead to maintenance costs, the loss of future development, and legal fees estimated to cost the town a net present value of $750,000.

The other two constraints are environmental. The operation of the well field must not cause the groundwater to drop below a specified minimum elevation beneath the waterway, and the wells must not be situated directly on the streambed. The Finite Difference Watershed Model checks whether a plan violates these constraints. Plans projected to violate the environmental constraints are ineligible for state and federal development grants, costing the town $250,000.

All of the above problem information is already written into the programs. As the user, your only task is to select your choice of well locations and pumping rates, run the programs, and record your results. Before setting out on your own, however, let's work through an example together.
Under the green “User Input Cells” heading in the “Cost Calculator” worksheet, the first four columns specify your chosen scenario. Enter the following values into those columns:

<table>
<thead>
<tr>
<th>Binary Variable, 1 or 0</th>
<th>Pumping Rate, 0 to 300</th>
<th>X-coordinate, 2 to 40</th>
<th>Y-coordinate, 2 to 40</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>9</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>150</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>0</td>
<td>100</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

As you enter these values into the spreadsheet you will notice that the watershed grid numbers are changing in the column to the right. (If your spreadsheet does not evaluate automatically as you enter values, you can either go to Tools→Options→Calculation and select “Automatic” or you can press F-9 when you would like the spreadsheet to evaluate.) When you have entered the above values, you will see the following:

<table>
<thead>
<tr>
<th>Well's Watershed Map Grid Number (needed as input for the finite difference model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>360</td>
</tr>
<tr>
<td>1428</td>
</tr>
<tr>
<td>584</td>
</tr>
<tr>
<td>809</td>
</tr>
<tr>
<td>43</td>
</tr>
</tbody>
</table>
With these grid map locations, you may now return to the “Finite Difference Watershed Example”. As described earlier, open the section titled “Creating a Pumping Scenario…” and enter the grid locations for the wells into the “wells” vector. You can copy and paste from Excel into Mathematica, but be sure to maintain the Mathematica syntax with commas separating each of the numbers.

The new pumping rates may be added in a similar fashion, with one important change. The binary variable (1 or 0) in the first worksheet column indicates whether your scenario will actually build the corresponding well. A “1” indicates that it is to be built, and a “0” indicates that it will not be built. If there is a “1” in the column for a well, then enter the pumping rate that is shown. If there is a zero, then make the pumping rate 0 in the finite difference model for that well. Again, be sure to watch the commas and the negative signs.

You should see a graphic that looks like this:
Now you must collect the model output. Open the “Groundwater Elevations at Well Sites” section. Type the new watershed map well grid numbers into the line of code as explained above and then press “Shift-Enter”. You should see the following:

\[
\begin{pmatrix}
79.5564 \\
79.6837 \\
80.2379 \\
79.6563 \\
81.5577
\end{pmatrix}
\]

These are the groundwater elevations at the various well sites. Open the “Any Constraint Violations?” section. You should see the word “True”. Now jump back over to the Excel worksheet and enter those values as shown below:

<table>
<thead>
<tr>
<th>Environmental Constraint Violation, T or F</th>
<th>Groundwater Elevation at Well</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>79.56</td>
</tr>
<tr>
<td></td>
<td>79.68</td>
</tr>
<tr>
<td></td>
<td>80.24</td>
</tr>
<tr>
<td></td>
<td>79.66</td>
</tr>
<tr>
<td></td>
<td>81.56</td>
</tr>
</tbody>
</table>

… and reporting results from the finite difference model

The worksheet now shows you the new total project cost of $1,331,624 (or something close to that if you rounded your numbers differently). This cost is also the fitness “score” for this solution. The lower the cost, the better the solution.
Exercises for the “Finite Difference Watershed Example” and the “Watershed Spreadsheet”

1. Run your own scenarios. You will need to decide how many wells to create, where they will be located, and how much each one will pump. After entering that information into the spreadsheet, enter the needed grid numbers and pumping rates into the finite difference model. After evaluating the model, find the groundwater elevations at your well sites, and check to see if you have violated the environmental constraint. Enter those finite difference model results into the spreadsheet.

   When you are satisfied with your best scenario, record the X-Y well coordinates, the pumping rates, and the total cost. Bragging rights may be at stake.

2. How did your mental process for creating your first scenario and then refining any subsequent scenarios differ from the process that a genetic algorithm would follow? You probably looked at the watershed map and made some considered choices. Discuss how that approach might be both a strength in your method and also a weakness.
Running the “Genetic Algorithm Watershed Program”

You have already learned just about everything you need to know to run this program. The codes from the “Genetic Algorithm Example” and the “Finite Difference Watershed Model” have been combined here into a single program. Some commentary and much of the illustrative graphics have been stripped away to make the program more streamlined, but the general format is basically unchanged.

Open the file and evaluate the notebook. In the section titled “The Problem” you will find the same problem statement that you encountered previously. The “Finite Difference Watershed Model” section contains the same model you saw before, but without all the pictures. In the “Representation” section you will find a description of how the problem variables (number of wells, pumping rates, X and Y coordinates) have been coded as chromosomes and genes. The “Recombination” section is essentially unchanged.

The “Mutation” section has some added details on how mutations which take individuals outside the physical limits of the space are handled. The remaining four sections are either unchanged or of optional interest to those who just want to know what is in there. The only other change is found when you press the “Set Parameters” button. Messrs. Madoff and Merdle have been replaced by new penalties which are appropriate to this problem.

A word of warning: This program has been written for ease of understanding and problem adaptation. It has a lot of calculations to perform. It is not very speedy. That is one of the downsides to evolutionary computation in general. It can be much slower than many of the other approaches. A problem-tailored linear program could potentially solve a problem like this in a fraction of the time. If you plan to conduct a large-scale “run”, I suggest that you let Mathematica operate in the background on your computer while you do other work or take a nap.
An audible indicator has been programmed to sound when the evaluation is complete, alerting the user that the results are ready. If the audible indicator is not wanted, it can be muted or the user may open the “Output” section and delete the 3 lines of code starting with “EmitSound…”, “Pause…”, and “Speak…” Alternatively, the user may want to customize the sounds. Adding your own name to the spoken message is a nice touch.

If you need to abort an evaluation, you may press “Alt + .” or click on “Abort Evaluation” under the “Evaluation” tab. And remember that large populations and lots of generations may not guarantee a good solution. There are other parameters to consider. If you want to “see” a solution graphically, just enter it into the “Finite Difference Watershed Example”.

There are on the order of $10^{30}$ different solutions to this problem. Suppose your computer was really fast and you could evaluate 1 million solutions every second. Then you would be able to check every possible solution in about a million times the age of the universe. Alternatively, you could use a method such as a genetic algorithm to search among the possibilities – finding a good solution instead of the best, but also shaving a few epochs off your processing time.

**A Final Word about Models**

As the saying goes, there is no free lunch. While genetic algorithms possess attributes which make them attractive for addressing certain problems, there will always be trade-offs when a model is used to solve a problem. If a genetic algorithm saves the programmer time in creating the model, it may cost the operator time when running the model. If a linear model allows for faster computation, it may also require approximations to fit the problem into a linear framework. Some methods can yield a guaranteed optimal solution; while others, such as genetic algorithms, only provide the best solution found so far. And with all models, it is important to remember that they are ultimately simplifications of reality. Simplified models always yield simplified solutions.
Exercises for the “Genetic Algorithm Watershed Program”

1. Set the parameters as you would have them, and run the program. When deciding on the parameter values, you might consider the following:

   » A good solution could potentially remain in the population indefinitely. Mutation and crossover operate on the new “offspring”. Large values will explore aggressively outward from the parent population. Smaller values will lead to more clustering and refinement around the best solutions.

   » The financial penalties for violating the water supply or the environmental constraints are fixed amounts in the problem statement. They are given as variable parameters in the program, however. If you suspect that you are not getting the best possible solutions to the problem, consider changing the penalties as a means to guide the population in the desired direction.

What is your best solution? Record your parameter settings, the complete solution details, and the elapsed evaluation time. You can copy and paste from Mathematica. To copy, select the cell(s) you want and then use Edit → CopyAs → Bitmap if you wish to retain the original formatting and graphics.

2. How does the genetic algorithm’s best solution compare with your best manual solution? What are some factors that make a problem well-suited for genetic algorithms?

3. How could this genetic algorithm be improved? What are some changes that would make it more efficient? More precise? Is the binary gene on each well chromosome a good idea?
Suggestions for Further Reading

Although I did not refer directly to any sources in writing the code for these learning module programs, the following books were essential to me in understanding how a genetic algorithm functions, how one is structured, and how to create a finite difference model:

Gives a readable and accessible introduction to the subject. There is little code in this book. The general concepts and strategies for implementing them are the main focus.

Goes into more depth than the Eiben book, but still contains sections that are accessible to the beginner. The most striking feature of the book is the multitude of illustrations in color and black-and-white showing evolutionary computation at work.

Definitely dated. My copy arrived with a giant floppy disk inside the back cover. Covers a wide range of numerical methods and includes a very short chapter on finite difference approximation for systems of partial differential equations. A newer version of Chapra’s book (2008) is available, but most of the material on finite differences has been removed.
Appendix B: “Genetic Algorithm Example” Code
Appendix C: “Finite Difference Watershed Example” Code
Appendix D: “Watershed Spreadsheet”

User Input Cells …

<table>
<thead>
<tr>
<th>Well #</th>
<th>Binary Variable, 1 or 0</th>
<th>Pumping Rate, 0 to 300</th>
<th>X, 2 to 40</th>
<th>Y, 2 to 40</th>
<th>Environmental Constraint Violation, T or F</th>
<th>Groundwater Elevation at Well</th>
<th>Well's Watershed Map Grid Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>200</td>
<td>8</td>
<td>34</td>
<td>F</td>
<td>79.56</td>
<td>321</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>200</td>
<td>13</td>
<td>11</td>
<td></td>
<td>80.06</td>
<td>503</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>50</td>
<td>22</td>
<td>16</td>
<td></td>
<td>80.66</td>
<td>877</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>100</td>
<td>36</td>
<td>6</td>
<td></td>
<td>80.29</td>
<td>1441</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>100</td>
<td>36</td>
<td>31</td>
<td></td>
<td>80.54</td>
<td>1466</td>
</tr>
</tbody>
</table>

Some Constants found in the Problem Statement

- Minimum Total Yield: 850
- Maximum Well Size: 275
- Well Drilling Cost: 70000
- Town Elevation: 100
- Cost Adj. Factor: 0.01
- Environmental Penalty: 250000
- Insufficient Yield Penalty: 750000

Well 1 Cost: 79839
Well 2 Cost: 84919
Well 3 Cost: 73491
Well 4 Cost: 78614
Well 5 Cost: 77397
Yield Penalty: 750000
Environmental Penalty: 0

Total Project Cost, $: 1144260
Appendix E: “Genetic Algorithm Watershed Program” Code
Appendix F: “Genetic Algorithm Watershed Program” Sample Solution

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Current</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size (even number)</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Number of Generations</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Mutation Distance</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Insufficient Yield Penalty</td>
<td>1 500 000</td>
<td>1 500 000</td>
</tr>
<tr>
<td>Environmental Violation Penalty</td>
<td>500 000</td>
<td>500 000</td>
</tr>
</tbody>
</table>

Best Solution:

\[
\begin{pmatrix}
0 & 275 & 39 & 36 \\
1 & 113 & 8 & 14 \\
1 & 215 & 30 & 28 \\
1 & 275 & 10 & 30 \\
1 & 251 & 32 & 37 \\
\end{pmatrix}
\]

Total Cost: 342 697.
Environmental Penalty: False
Yield Penalty: False
Number of Wells: 4
Total Water Yield: 854

Total evaluation time (minutes): 99.3