34 VBA PROGRAMMING FOR LANDSCAPE ANALYSIS

In collaboration with Robert Rohr

Objectives

- Develop a hypothetical raster-type landscape on a spreadsheet.
- Write Visual Basic for Applications code to analyze patch characteristics of the landscape.
- Write VBA code to analyze edge and core characteristics of the landscape.
- Develop spreadsheet formulas to calculate basic landscape statistics.

Suggested Preliminary Exercise: Landscape Ecology: Quantifying Landscape Pattern

INTRODUCTION

Take a look around as you walk or drive around the countryside surrounding your home. You may see prairie habitat interspersed with farm fields, or wetland habitats embedded within a sea of forest, or a mosaic of homes, buildings, and parks. No matter where you live, you can see a *patchwork* of many different habitats. Plants and animals live in this mosaic, and the amount and arrangement of the various habitats can affect the number of individuals, population growth rates, and evolution of various species.

For example, in the midwestern United States, forest songbirds that nest in small habitat patches surrounded by farmland often have low nesting densities, high rates of nest predation, and high rates of brood parasitism (Robinson et al. 1995). In contrast, populations nesting in much larger forest patches where there is less farmland have high population densities and low rates of predation and parasitism. What role does farmland play in creating to these patterns? Does the amount of farmland have anything to do with it? Would nesting densities differ if a corridor of forest were maintained through the farmland? If so, where should the corridor be positioned to benefit the songbird population? A landscape ecologist would be interested in answering these questions.

Landscape ecology is the study of the *amount* and *arrangement* of habitats and how these affect ecological patterns and processes (Turner 1989). **Ecological patterns** include the distribution and abundance of organisms. (Where are they found? How many are there?) Processes include population growth, dispersal, and evolutionary change (Are populations increasing, decreasing, or stable? Are individ-

uals able to disperse from one habitat to another?). Although we often think of landscapes on a scale of tens of kilometers, the study of landscape ecology can take place at any scale. The dynamics of a population of mites that occur in a few square inches of soil may be driven by the amount and arrangement of different soil particles!

As Wiens (1997) stated, "The underlying premise of landscape ecology is that the explicit composition and spatial form of a landscape mosaic affect ecological systems in ways that would be different if the mosaic composition or arrangement were different."

Until recently, the majority of ecological theories on population dynamics, predator-prey interactions, competition, and life-history variation assumed that habitats were either homogeneous, or implied that heterogeneity of habitats played a small role in population dynamics. In recent years, however, ecologists have realized that we need to understand how spatial patterns affect population dynamics in order to more accurately predict population responses to various situations. As a result, the field of landscape ecology (or **spatial ecology**) has grown rapidly, and today conservation biologists, land planners, wildlife managers, and many empirical and theoretical scientists routinely do landscape analyses.

How are landscapes quantified? Dig out a quad map such as those used in Exercise 30 and you can quantify the amount of a given habitat (forest) versus nonhabitat (nonforest). If you find, for example, that the quad map contains 40% forest cover and 60% nonforest, you have generated a landscape statistic that may be useful in your studies. (Note that this statistic does not tell you how the forest is distributed—only that about 40% of the land on the map is forest.) Landscape ecologists who are interested in coarse-grain habitat patterns often use aerial photographs or satellite images to study a large area of interest, and then classify and quantify the different habitats with the aid of a **Geographic Information System (GIS)** and appropriate software.

One kind of GIS is a raster-based system in which a picture (such as a satellite photo) is divided into many "pixels" or "cells," each of which depicts a certain amount of the earth's habitat as viewed from outer space. A pixel, for example, might consist of 100 square meters of habitat—a plot of land 10 meters on a side. Each pixel is then assigned to a habitat type. If the majority of a pixel consists of forest habitat, the cell or pixel is classified as forest. This process is done for each and every pixel in the landscape. Once pixels have been classified, the landscape can be described in terms of amount and arrangement of the various habitats. This information is useful for landscape ecologists, who focus on how landscape patterns affect ecological patterns and processes.

PROCEDURES

The goal of this exercise is to introduce you to a kind of computer programming called Visual Basic for Applications (VBA for short). This exercise is a bit different from others in this book in that it is not a model *per se*, but instead uses a spreadsheet as a tool to quantify landscape pattern. You'll write a VBA program that will turn your spreadsheet into a GIS of sorts, with the ability to analyze landscape statistics. You'll be able to "design" a landscape that consists of forest and nonforest habitat. Then you will generate statistics including metrics such as average patch size, amount of edge habitat, and amount of core habitat. You have already computed these metrics by hand in Exercise 30. However, as you probably learned, these computations can be tedious and time-consuming, and hence it is very difficult to compare different landscapes quickly. The VBA code will allow you to rapidly compute metrics for any landscape you design.

Let's start by examining Figure 1. The VBA program you will write will take the information in cells C25–L34 and assign a patch number to each of the forested cells (cells C10–L19). See if you can follow the assignment of the patch numbers to the each of the forested cells in the landscape; it should make sense to you. Once each cell has been

	B	C	D	E	F	G	H	I	J	K	L	M
8		**Analyzed data (Plot_Range)**										
9												
10												
11			1					2				
12				3				2				
13			3	3				2			4	
14								2	2	2		
15												
16				5								
17			6				7	7				
18							7	7				
19												
20												
21												
22												
23		**Raw data (Raw_Range)**										
24												
25												
26			f					f				
27				f				f				
28			f	f				f			f	
29								f	f	f		
30												
31				f								
32			f				f	f				
33							f	f				
34												
35												

Figure 1 The landscape represented here consists of a 100-cell matrix, each cell of which represents a 10 m × 10 m plot. The raw data is stored in cells C25–L34. Each pixel is assigned an "f" if it consists of forest habitat and is left blank if it consists of nonforest habitat. Forested pixels are shaded; nonforested pixels are white. If we assume that a patch consists of groups of cells that adjoin each other in one of the four cardinal directions, 7 patches are shown. The largest patch is 6 cells in size, or 600 m^2. This patch has 14 edges, which together make up 140 meters of edge habitat. The smallest patch is 1 cell (100 m^2). None of the patches has **core cells** (i.e., cells that are surrounded by forest, with no adjoining nonforest habitat). Patches that consist of a single cell have a small **shape index** because they resemble a square; the patch with 6 cells has a large shape index because it is more convoluted.

given a patch number, you will use simple spreadsheet functions (MIN, MAX, AVERAGE, STDEV, etc.) to calculate various landscape statistics.

The beauty of the spreadsheet is that you can manipulate your landscape in any way you choose, which in turn will allow you to understand what the different landscape metrics convey (and what they don't convey!). **Save your spreadsheet and VBA code! You will use it in the next exercise** ("Neutral Landscapes and Connectivity").

As always, save your work frequently to disk.

INSTRUCTIONS

ANNOTATION

A. Set up the landscape model.

1. Open a new spreadsheet as shown in Figure 2. Your landscape, labeled "Raw Data," consists of 100 cells, located in cells C25–L34.

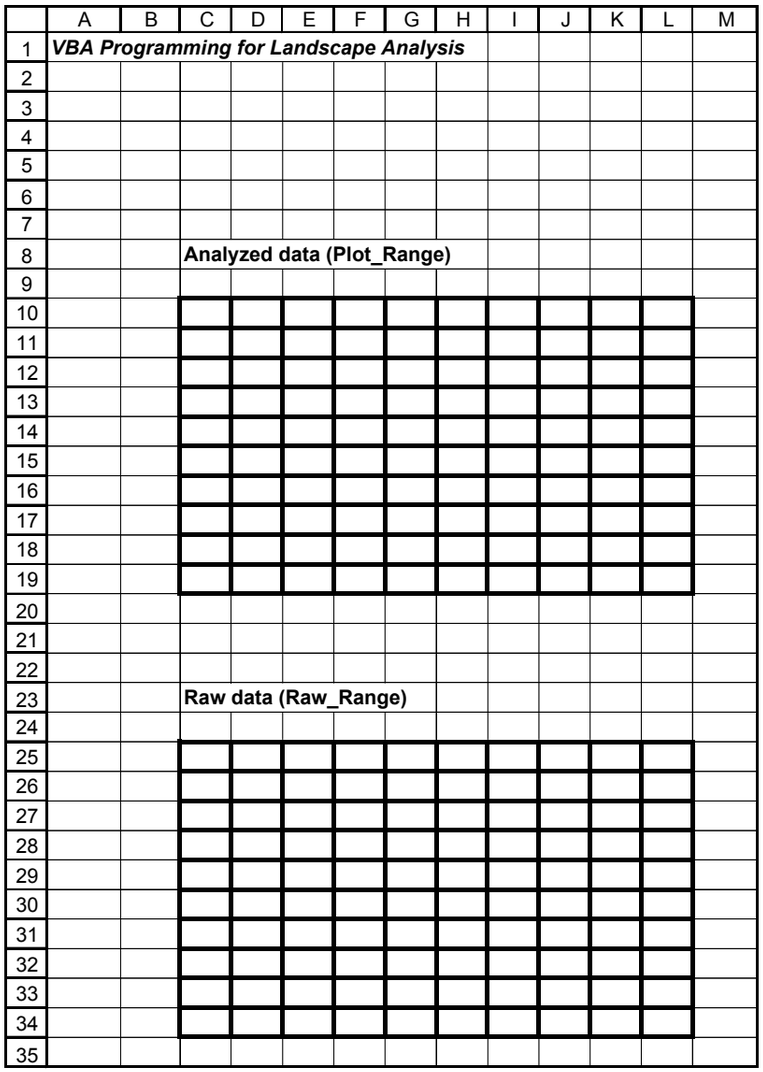

Figure 2

2. Randomly select 20 cells to be forested landscape.

Choose 20 cells *at random* within your landscape and enter the letter "f" in those cells. Leave the nonforested cells blank.

3. Conditionally format the forested cells to be shaded green.

Highlight cells C25–L34, then go to Format | Conditional Format. A dialog box will appear as shown in Figure 3. Set Condition 1 as "cell value equal to f," then click Format, and then the Patterns option and select a green shade. Your forested cells should be formatted automatically as you change your landscape.

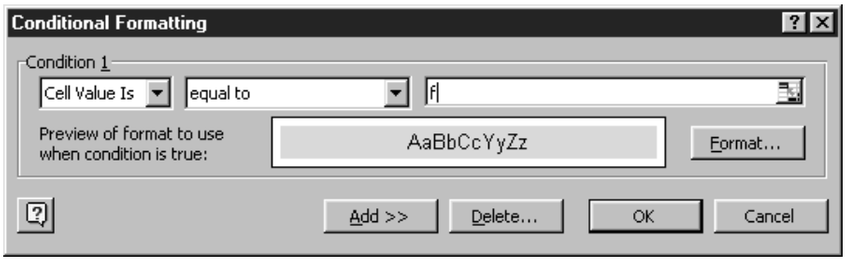

Figure 3

4. Save your work.

B. Set up the macro to analyze patch statistics.

1. Go to Tools | Macro | Record New Macro. Type in the name of the macro and a shortcut key.

The dialog box shown in Figure 4 will will appear. The name of the macro is AnalyzePatch, and the shortcut key assigned is <Control><Shift><a>. Once the name and shortcut key are entered, click OK.

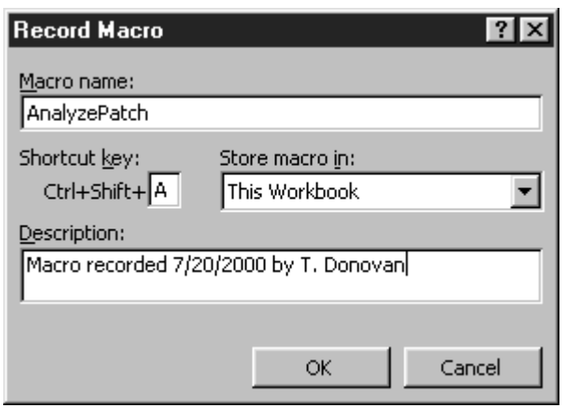

Figure 4

2. Stop recording.

If the Stop Recording toolbar appears, click on the Stop button. Otherwise, go to Tools | Macro | Stop Recording.

3. Edit the macro.

Go to Tools | Macro | Macros, select your macro (AnalyzePatch), and click the Edit button. You should see VBA code similar to that in Figure 5. Excel "wrote" this VBA code as a result of steps 1 and 2. The lines that begin with a single quote (') are *comments*; they are not actually read as instructions for the computer, but are simply statements made by the programmer (you) to explain the code you've written. The comments above indicate the name of the macro, who recorded the macro, and what the keyboard shortcut is for running the program. The program begins with the code `Sub AnalyzePatch()`, and ends with the code `End Sub`. You will type in VBA code in between these statements (you'll also type in comments to help you follow what the code means).

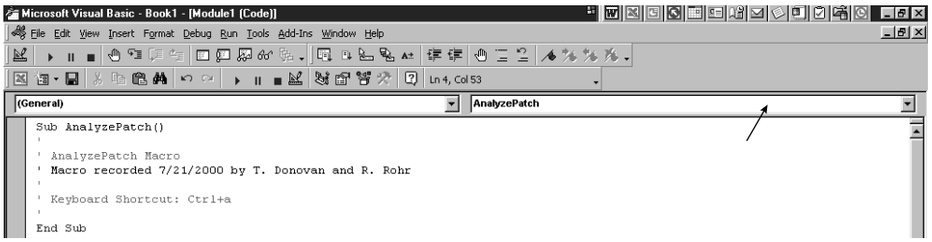

Figure 5

4. Select the Declarations window, then type in the words `Option Explicit` and press return.

Note that here is a drop-down window in the upper right corner of the window in Figure 5 (arrow). Currently the window reads "AnalyzePatch." Select the drop-down arrow to the right of the words AnalyzePatch and you'll find another option called Declarations.
Your cursor should automatically move to the top of the page when you select the Declarations window. Type in the words `Option Explicit`
Your code should now look like this:

```
Option Explicit
_____________________________-
Sub AnalyzePatch()
```

5. Move your cursor one line above the `End Sub` code (delete the single quote), and continue with the next section.

The Explicit option lets you explicitly define the variables in your program (i.e., define the variables at the beginning of the program rather than implicitly defining them as they are encountered in the program). You'll have variables with names such as `Patch_Count` and `New_Neighbor_Found`. We'll explain what each of these variables is meant to accomplish in the next section. The `Explicit` command requires that you explicitly declare *all* the variables you will use in your program at the beginning; VBA will not recognize any variables not explicitly declared. We use the Explicit command in this program to help point out typos—which hopefully will help you avoid headaches in typing the program!

6. Save your work.

C. Type in Visual Basic code.

1. Type in the code to the right *exactly* as it is written (including indentations). Save your work!

The indentations or tabs are not required for proper execution of the code, but will help you follow the structure of the code. The codes in bold font are comments; they briefly describe what the code that follows it is intended to accomplish. The single "underline" character, when it appears at the end of a line, tells the VBA compiler that the code continues onto the following line. Any errors you make along the way will appear in red on your screen. We will explain each part of the code in Section D.

```
'Define Variables
Dim MySheet As Object
Dim Plot_Cell As Object

Const Plot_Range As String = "C10:L19"
Const Raw_Range As String = "C25:L34"

Dim Patch_Count As Integer
Dim New_Neighbor_Found As Boolean
Dim No_Remaining_Forested_Cells As Boolean
Dim No_New_Neighbors As Boolean
Dim First_Cell_This_Patch As Boolean
```

```
Dim i As Integer, EdgeCount As Integer, NeighborCount As
Integer, _
CoreCount As Integer

Set MySheet = Worksheets("Sheet1")

'Clear out old edge and core count data
With MySheet
  .Range("B100", .Range("B100").End(xlDown)).EntireRow.Delete
End With

'Copy Raw Data into analysis range
MySheet.Range(Raw_Range).Copy
MySheet.Range(Plot_Range).PasteSpecial Paste:=xlValues, _
Operation:=xlNone, SkipBlanks:= _
  False, Transpose:=False

'Perform Nested Patch Loops

'Starting Analysis Assumptions
Patch_Count = 0
No_Remaining_Forested_Cells = False

'Outer Loop — Repeats once per patch
Do Until No_Remaining_Forested_Cells = True

  'Reset variables for new patch search
  Patch_Count = Patch_Count + 1
  No_New_Neighbors = False
  First_Cell_This_Patch = False

  'Inner Loop — Repeats until no more forested neighbors are
  'found adjacent to the current patch.
  Do Until No_New_Neighbors = True

    New_Neighbor_Found = False

    'Performs one complete pass through the grid looking
    'for forested cells
    For Each Plot_Cell In Range(Plot_Range)

      'Identifies the first forested cell in a patch and
      'forces all future searches for this patch to be
      'neighbor comparisons
      If UCase(Plot_Cell.Value) = "F" And _
      First_Cell_This_Patch = False Then
        Plot_Cell.Value = Patch_Count
        First_Cell_This_Patch = True
        New_Neighbor_Found = True
      End If

      'Identifies and marks neighbor cells of the current
      patch
      If UCase(Plot_Cell.Value) = "F" And _
        (Plot_Cell.Offset(-1, 0).Value = Patch_Count Or _
        Plot_Cell.Offset(0, 1).Value = Patch_Count Or _
```

```
        Plot_Cell.Offset(1, 0).Value = Patch_Count Or _
        Plot_Cell.Offset(0, -1).Value = Patch_Count) Then
          Plot_Cell.Value = Patch_Count
          New_Neighbor_Found = True
      End If

    Next Plot_Cell

    If New_Neighbor_Found = False Then No_New_Neighbors =
True

  Loop

  If First_Cell_This_Patch = False Then
  No_Remaining_Forested_Cells = True
  End If

Loop

'Start Edge and Core Count Analysis
With MySheet.Range("C99")

  For i = 1 To Patch_Count - 1

    .Offset(i, 0).Value = "Patch " & i
    'Count Patch Edge sections by subtracting
    'patch neighbors from 4

    EdgeCount = 0
    CoreCount = 0
    For Each Plot_Cell In Range(Plot_Range)
      If Plot_Cell.Value = i Then
        NeighborCount = -1 * ((Plot_Cell.Offset(-1, 0).Value
        = i) _
          + (Plot_Cell.Offset(0, 1).Value = i) _
          + (Plot_Cell.Offset(1, 0).Value = i) _
          + (Plot_Cell.Offset(0, -1).Value = i))
        EdgeCount = EdgeCount + (4 - NeighborCount)
      End If

      'Count Core cells by tallying cells with 4 neighbors
      If Plot_Cell.Value = i And NeighborCount = 4 Then
        CoreCount = CoreCount + 1
      End If

    Next Plot_Cell
    .Offset(i, -1).Value = EdgeCount
    .Offset(i, 1).Value = CoreCount

  Next i

End With

Calculate

'Reset Cursor to upper left corner
```

```
MySheet.Range("C25").Select

Set MySheet = Nothing

End Sub
```

D. VBA Patch Analysis code explanations.

1. 'Define variables

The first section of the code defines the variables used in the program. The value of a variable can change as the program runs. When you define a variable in VBA, the computer allocates some memory space to store information associated with that variable. For example, we will define the variable `Patch_Count` and use it to assign each of the forested cells in our landscape a number that identifies which patch the forested cells belong to. As the program runs, the `Patch_Count` variable will take on different values (1, 2, 3, etc.) to identify unique patches in the landscape. A variable name is defined by typing in `Dim` (short for dimension), then the variable name, and then a description of the type of variable it is.
We will use the following types of variables in our program: **objects**, **strings**, **Boolean variables**, and **integers**.

`MySheet` and `Plot_Cell` are defined as **objects** in your code. An object in VBA language is something that has properties or methods associated with it. For example, a spreadsheet chart is an object that can be described by several properties such as size, font, and color, and can be told to recalculate (a method) or print. `MySheet` will refer to the spreadsheet that contains the landscape information you want to analyze. `Plot_Cell` will refer to cells within the landscape.

`Plot_Range` and `Raw_Range` are defined as **strings**. A string in VBA language is text data. If you want to reference a group of cells such as A1–A21, the address (A1:A21) consists of a combination of letters and numbers, not just numbers, so a strictly numerical variable would not interpret the address correctly. You'll notice that instead of the `Dim` code, `Const` is used in front of `Plot_Range` and `Raw_Range`. `Const` stands for constant, which is similar to a variable. Constants, however, have values that do not change as the program is running. The constant `Raw_Range` holds the address of the raw landscape data (C25:L34). These are the cells that represent our hypothetical landscape, and each either has the letter f or is blank. The information in `Raw_Range` will be copied up to the analysis range—the group of cells we will use to analyze the raw landscape data. The constant `Plot_Range` holds the address of the analysis range (C10:L19). The goal of the first part of the VBA program is to "reassign" each forested cell in the `Plot_Range` to a number that indicates which patch it belongs to (i.e., turn the f's into 1, 2, 3, etc.).

The variable `Patch_Count` is an **integer**, or whole number. The program will change the f cells in the `Plot_Range` into numbers that identify which patch a particular cell belongs to. As the program runs, `Patch_Count` will be updated so that it always holds the number of the current patch under investigation. The variables `i`, `EdgeCount`, `NeighborCount`, `CoreCount`, and `CoreCell` are integer variables used in the edge and core analysis.

The variables `New_Neighbor_Found`, `No_Remaining_Forested_Cells`, `No_New_Neighbors`, and `First_Cell_This_Patch` are **Boolean variables**. VBA recognizes that these variables can take on only one of two values: `True` or `False`. We will describe what each of these variables does as they are encountered in the code.

The last line in this section is the `Set MySheet` code. The object `MySheet` (defined

earlier) is set, or linked, to Sheet1 in your program; this is the sheet that contains your landscape information.

2. 'Clear out old edge and core count data

The next section of code functions to clear out any analyses related to edge and core habitat that were previously run. This allows you to have a "blank slate" for future analyses. The code uses the `With … End With` structure, which allows an object (in this case `MySheet`) to be manipulated without having to specify all of the object's properties over and over again. It saves us typing time and saves the computer processing time. In this case, we want the program to go to the range defined by cell B100 and all the cells below B100 that contain data, as specified by `.End(xlDown)`, and then delete all the rows in that range.

3. 'Copy raw data into analysis range

The next section copies the `Raw_Range` data (located in cells C25–L34) and pastes it into the analysis range, which is called `Plot_Range` (located in cells C10–L19). After this step, the `Plot_Range`, like the `Raw_Range`, also contains cells that either are blank or have the value f.

4. 'Perform nested patch loops

Broadly speaking, there are two loops in this section: an outer loop and an inner loop. The outer loop mostly keeps track of the number for the habitat patch currently being analyzed. The inner loop finds the first cell in a new patch and identifies all the forested cells that belong to that patch, assigning the current value of `Patch_Count` to those cells. When all of the forested cells within that patch have been located (when the f has been changed to a number in each case), the inner loop is broken and the code returns to the outer loop. Patches that have already been located now have a number assigned to each cell, whereas unassigned patches will still have the letter f associated with them. When all of the patches in the landscape have been identified (there are no more f's in the landscape), the outer loop is broken and the analysis is complete.

5. 'Starting analysis assumptions

Before we start the outer loop, we will set some variables to show that no patches have been assigned numbers yet. First, we will set the variable `Patch_Count` to 0, which indicates that, to this point, no patches have been found in the landscape. The variable `No_Remaining_Forested_Cells` is a Boolean variable that indicates whether or not any forested cells are in the landscape. We will set `No_Remaining_Forested_Cells` to `False` to indicate that we expect to find some forested cells in the landscape.

6. 'Outer loop – Repeats once per patch

Now we get into the meat of the program. Both the outer and inner loops use the `Do Until … Loop` structure. This pair of commands tells the computer to do everything between the words `Do` and `Loop` until the condition given after `Until` is met. The outer loop is repeated until the variable `No_Remaining_Forested_Cells` is set to `True`. This variable will be set to `True` when the program makes a complete pass through every cell in the `Plot_Range` (which it does each time the inner loop runs) and finds no cells that contain the letter f. If `No_Remaining_Forested_Cells` is not `True` (i.e., if it is `False`), there may still be some f's in the landscape, so the outer loop runs again. The `Loop` command at the end of the patch analysis program (just before the edge analysis program) is responsible for this looping. Note that the `Loop` command for the inner loop occurs before this; the pattern of indentation should help you keep track of which `Loop` goes with which `Do`.

When the outer loop is run, three variables are reset. `Patch_Count` is now set to `Patch_Count + 1`. Earlier, we set `Patch_Count` to 0. In the very first "trip" through the outer loop, the `Patch_Count` becomes 0 + 1 = 1. The inner loop will continue to work on patch 1 until all of the f cells that make up patch 1 have been assigned the number 1, at which point the inner loop is terminated and the outer loop is run again. At that time, `Patch_Count` becomes 2 (or 1 + 1), and so on. At the beginning of the outer loop, we also set the variable `No_New_Neighbors` to `False` and the variable

`First_Cell_This_Patch` to `False`, meaning that we have not yet found the first cell in a brand new patch, and that new neighbors are yet to be discovered. Thus, when we start working on patch 2, we assume that the first cell in patch 2 has not yet been located (`First_Cell_This_Patch = False`) and that neighbors for patch 2 cells are out there to be found (`No_New_Neighbors = False`).

7. 'Inner loop

The inner loop starts to search for the first cell in a new patch, assigns the current patch number to the first cell that is located, and "locates" all of the other f cells that are also part of that same patch. The inner loop runs over and over again searching for unassigned members of the patch until the variable `No_New_Neighbors` is set to `True`, which indicates that none of cells with the letter f in the `Plot_Range` is adjacent to a cell with the number of the current patch. Before the inner loop is run, the variable `New_Neighbor_Found` is set to `False`, which means that new neighbors have not yet been located. The pair of commands `For Each Plot_Cell In Range(Plot_Range)` and `Next Plot_Cell` tell the program to look at each cell in the `Plot_Range` every time the inner loop is run. If the variable `New_Neighbor_Found` is still `False` at the end of the inner loop, then the program has checked every cell in the `Plot_Range` and determined that there are no more cells in the current patch. It therefore sets the variable `No_New_Neighbors` to `True`, and the inner loop is broken.

There are two `If … End If` blocks in the inner loop. The first block identifies the first forested cell in a new patch. The code basically says if the cell has a value of f (upper or lower case is fine) and the variable `First_Cell_This_Patch` is still `False` (i.e., if the first cell in the patch has not been located yet), then we've found the first cell in a brand new patch. Then, the program assigns its value to the current patch count number (it changes the f to the number that is currently stored in the variable `Patch_Count`). The program also sets `First_Cell_This_Patch` to `True` to indicate that the first cell in the patch has now been located, and `New_Neighbor_Found` to `True` to indicate that new neighbors are being found (the first cell is considered to be its own neighbor). Note that if another forested cell is located, the variable `First_Cell_This_Patch` is now `True`, so the first `If … End If` block is not carried out and the code kicks down to the second block.

The second `If … End If` block identifies and marks all the cells that are part of the current patch. The code in this block will run only if the cell under investigation contains the letter f and if that cell is adjacent to a cell that contains the number currently stored in the variable `Patch_Count`. The `Offset` code tells the program which cells to check for the number of the current patch. The code `Offset (-1, 0)` searches the cell above to see if it has the current patch count; `Offset (0, 1)` searches the cell to the right; `Offset (1, 0)` searches the cell below, and `Offset (0, -1)` searches the cell to the left. If any of the neighboring cells have the current patch count number assigned to it, the f is flipped to the current `Patch_Count` value. If a new neighbor is found, the `New_Neighbor_Found` variable is set to `True`, and the inner loop is repeated again to continue searching for new neighbors (not all neighbors will necessarily be located in a single pass through the inner loop). Once a complete pass through the `Plot_Range` occurs and no new neighbors are found, the patch is complete and the program leaves the inner loop. The two `If` blocks are repeated for each cell in the landscape. The code `If New_Neighbor_Found = False Then No_New_Neighbors = True` identifies when the program can exit the inner loop.

The outer loop is terminated by the code `If First_Cell_This_Patch = False Then No_Remaining_Forested_Cells = True`. This basically says that when the first cell in a new patch cannot be located (i.e., when the program has made a complete pass through each cell in the `Plot_Range` and found no cells that contain the letter f), the variable `No_Remaining_Forested_Cells` should be set to `True`, which in

turn terminates the outer loop. Your patch analysis is basically completed; cells C5–L14 no longer contain f values; instead, each f was replaced by a patch number. You are now ready to calculate how much edge each patch has, which is accomplished in the next section.

E. VBA Edge and Core Analysis code.

1. 'Start Edge and Core Count Analysis

The edge analysis starts by writing in the words Patch 1 in cell C100, Patch 2 in cell C101, Patch 3 in cell C102, and so forth, until all of the patches in the `Plot_Range` have been identified. In the code `For i = 1 To Patch_Count - 1` the variable `i` gives the number of the `Patch_Count`, and takes on new values for each new patch in the landscape. Thus, if there are four patches in the landscape, `i` takes on the values 1 through 4. (You might be wondering why the code says `For i = 1 To Patch_Count - 1`, instead of `For i = 1 to Patch_Count`. This is because before the patch analysis program is terminated, the `Patch_Count` is already assigned the next new number, although none of the cells will be assigned to this number because there are no more forested cells in the landscape. At the end of the patch analysis program, therefore, `Patch_Count` holds a number one larger than the actual number of patches in the landscape.) The program will analyze the amount of edge and the area of core for each patch in the landscape. The amount of edge will be listed to the left of the patch identifier, and the amount of core habitat will be listed to the right of the patch identifier. For example, the amount of edge associated with patch 1 will be given in cell B100, and the amount of core habitat in patch 1 will be given in cell D100.

2. 'Count Patch Edge sections by subtracting patch neighbors from 4

The code `With MySheet.Range("C99")` selects cell C99, and the code `.Offset(i, 0).Value = "Patch " & i` offsets cell C99 down `i` cells, and then changes the value of that cell to the word `Patch` followed by the number held in the variable `i`. For example, when `i` = 1, the value `Patch 1` is assigned to cell C100; when `i` = 2, the value `Patch 2` is assigned to cell C101; and so forth.

For each patch, the program starts by setting the variables `EdgeCount` and `CoreCount` to 0. As the program examines each cell in a patch, it will increase `EdgeCount` by the number of edges the cell has; `EdgeCount` will therefore eventually tell us the total number of edges a patch has. The program calculates the number of edges each cell has by counting the number of neighboring cells that also belong to the patch under investigation, storing this value in the variable `NeighborCount`, and subtracting this number from 4. For example, if a cell has three neighbors, `NeighborCount` is 3 and the program increases `EdgeCount` by a value of (4 – `NeighborCount`), or 1. This is accomplished with the code `EdgeCount = EdgeCount + (4 - NeighborCount).`

3. 'Count core cells by tallying cells with 4 neighbors

A cell is core habitat if it is surrounded by similar habitat in the four cardinal directions, that is, if it has four neighbors. Thus, if a cell's `NeighborCount` is 4, then it is a core habitat cell. The code `If Plot_Cell.Value = i And NeighborCount = 4 Then CoreCount = CoreCount + 1` increases the value of `CoreCount` by 1 for each cell of core habitat found in patch `i`.

4. Reset cursor and end macro.

The last code in the program resets your cursor to cell C20, and then sets MySheet = Nothing. This "nothing" code frees up memory used by the program and lets you start the whole analysis over with a clean slate.

5. Save your work.

F. Compute patch statistics.

1. Set up new column headings as shown in Figure 6.

	N	O	P	Q	R	S	T	U	V
6	Patch statistics								
7	Patch #	Area (m^2)	Area (m^2)	Edge (m)	Edge (m)	Core (m^2)	Core (m^2)	Shape	Fractal

Figure 6

2. In cells N8–N37, set up a linear series from 1 to 30. (You can extend it to 50 if you wish).

The statistics for each patch will be calculated under the cells headed "Patch statistics." Enter 1 in cell N8.
Enter the formula **=1+N8** in cell N9 and copy this formula down to cell N37.

3. In cells O8–O37, enter a formula to calculate the area of each patch in the landscape.

Enter the formula **=COUNTIF($C$10:$L$19,N8)*100** in cell O8 and copy this formula down to row O37.
The **COUNTIF** formula counts the number of cells in a range that meet a certain criteria. It has the syntax **COUNTIF(range,criteria)**. The O8 formula examines the cells in C10–L19 and counts the number of cells that contain the number 1 (which is listed in cell N8). Since each cell represents an area of 100 m^2, the formula multiplies the number of cells by 100.

4. In cells P8–P37, enter an **IF** formula to return a patch size only for those patches that actually exist on the landscape.

Enter the formula **=IF(O8>0,O8,"")** in cell P8 and copy the formula down to P37.
(We need to perform this step so that statistics like average patch size are correctly computed.) The P8 formula tells Excel to evaluate the contents of cell O8. If O8 is greater than 0, return the value currently in cell O8; otherwise return a blank cell (indicated by the double quotes).

5. Use the **INDIRECT** function to transfer the edge data from your VBA program over and up to cells Q8–Q37.

Enter the formula **=INDIRECT("B100")*10** in cell Q8. Enter analogous formulae to calculate edge for the remaining patches.
The **INDIRECT** formula in Q8 returns the contents of cell B100, which are the edge statistics associated with patch 1. (We need to use this function because the edge analyses are deleted each time the macro is run.) This number is multiplied by 10 to give the meters of edge present for each patch. Copy this formula down to row 37. You will have to manually adjust the formula in each cell to reflect the correct edge amount for a given patch. For example, we used the formula **=INDIRECT("B101")*100** in cell Q9.

6. In cells R8–R37, enter an **IF** formula to return landscape statistics for those patches that actually exist on the landscape.

Enter the formula **=IF(Q8>0,Q8,"")** in cell R8. Copy the formula down to cell R37.
Similar to step 4, this calculation is necessary so that we can correctly compute some landscape statistics later in the exercise.

7. Use the **INDIRECT** function to transfer the core data from your VBA program over and up to cells S8–S37.

Enter the formula **=INDIRECT("D100")*100** in cell S8 and fill in analogous formulae for the remaining patches.
As in step 5, you will need to manually change the formula to reflect the amount of core habitat for a given patch. The formula multiplies the number of core cells by 100 because each core cell represents 100 m^2 of the landscape.

8. In cells T8–T37, use the **IF** function in combination with the **AND** and **OR** functions to return a value only for those patches that actually exist on the landscape.

Enter the formula **=IF(AND(Q8>0,OR(S8=0,S8>0)),S8,"")** in cell T8. Copy the formula down to T37.
The catch here is that patches can exist and still have no core. The T8 formula specifies that cell Q8 must be greater than 0, **AND** S8 must be greater than **OR** equal to 0 to return the value in cell S8. Otherwise, the spreadsheet will return a blank cell (**""**).

9. Calculate the shape index in cells U8–U37.

Enter the formula **=IF(R8="","",(R8*0.25)/(SQRT(P8)))** in cell U8. Copy the formula down to U37.
We used an **IF** function so that the shape index is calculated only for those patches that actually exist on the landscape. The **shape index** for a patch is the perimeter (edge) of the patch (in meters), multiplied by 0.25 and then divided by the square root of the area (in m^2) of the patch. When a patch is square, the shape index is 1. As patches become more irregularly shaped, the shape index increases.

10. Calculate the fractal dimension of each patch in cellsV8–V37

Enter the formula **=IF(P8="","",(2*LN(0.25*R8))/(LN(P8)))** in cell V8. Copy the formula down to cell V37.
Again, we used an **IF** function so that the fractal dimension is calculated only for those patches that actually exist in the landscape. The fractal dimension is another shape statistic. It ranges between 1 and 2 and indicates how much a patch departs from a square in shape. A square patch has a fractal dimension of 1, whereas a very convoluted patch will have a fractal dimension close to 2.

11. Save your work.

G. Calculate landscape statistics.

1. Set up new headings as shown in Figure 7.

	X	Y	Z	AA
6	**Landscape statistics**			
7	Total area (m^2)			
8	Proportion forest			
9	Proportion core habitat			
10	Proportion nonforest			
11	Total edge			
12	Simpson's diversity index			
13	Shannon's diversity index			
14				
15	**Patch statistics**			
16	Number of patches			
17	Maximum patch size			
18	Minimum patch size			
19	Average patch size			
20	Std. patch size			
21	Patch size CV			
22	Average core area			
23	Std. core area			
24	Core area CV			
25	Average edge (m)			
26	Std. edge (m)			
27	Average shape index			
28	Std. shape index			

Figure 7

2. In cells AA7–AA28, write formulae to compute the landscape statistics. Reference your data in cells N–V, or the PatchPlot landscape in cells C10–L19.

We used the following formulae:

- AA7 Enter the number **10000** (to represent 10,000 m^2).
- AA8 **=1-AA10**
- AA9 **=SUM(S8:S37)/AA7**
- AA10 **=COUNTIF(C10:L19,"")*100/AA7**
- AA11 **=SUM(Q8:Q37**)
- AA12 **=1-(AA8^2+(AA10^2))**
- AA13 **=-(AA8*LN(AA8)+AA10*LN(AA10))**

- AA16 **=MAX(C10:L19)**
- AA17 **=MAX(P8:P37)**
- AA18 **=MIN(P8:P37)**
- AA19 **=AVERAGE(P8:P37)**
- AA20 **=STDEV(P8:P37)**
- AA21 **=(AA20/AA19)*100**
- AA22 **=AVERAGE(T8:T37)**
- AA23 **=STDEV(T8:T37)**
- AA24 **=(AA23/AA22)*100**
- AA25 **=AVERAGE(R8:R37)**
- AA26 **=STDEV(R8:R37)**
- AA27 **=AVERAGE(U8:U37)**
- AA28 **=STDEV(U8:U37)**

3. Save your work.

QUESTIONS

1. Choose 3 or 4 landscape metrics (e.g., edge length) and run your analyses on 10 different landscape configurations. Do the metrics adequately give you a "picture" of what the landscape looks like? Which metrics give you a good visual picture of the landscape? What are the shortcomings of the various metrics regarding their ability to provide a good visual picture of the landscape?
2. Average patch size is a statistic that is commonly used to describe landscapes. Design two landscapes, each consisting of just two patches. Let one landscape have one very large and one very small patch, and the other landscape have two equal sized patches. Does the average patch size metric provide an adequate visual picture of the landscape? What other metric is needed to interpret how large the various patches are? Can you think of a way to improve the average patch size statistic (e.g., weighting patches by their size)?
3. Shape index and fractal dimension both provide information about how a patch is shaped. How do these metrics function?
4. What information do the diversity indices convey? Arrange your landscapes in various fashions to demonstrate how the statistics describe various kinds of landscapes.
5. Are there other indices that might be useful (but not programmed) in terms of describing a landscape? Make up two hypothetical landscapes that are similar in many respects (amount of forest, edge, average patch size, etc.), but differ in terms of location of the patches. How might location of patches be incorporated into your program?

LITERATURE CITED

Robinson, S. K., F. R. Thompson III, T. M. Donovan, D. Whitehead and J. Faaborg. 1995. Regional forest fragmentation and the nesting success of birds. *Science* 267: 1987–1990.

Turner, M. G. 1989. Landscape ecology: The effect of pattern on process. *Annual Review of Ecology and Systematics* 4: 21–30.

Wiens, J. 1997. Metapopulation dynamics and landscape ecology. *In* I. Hanski and M. Gilpin (eds.), *Metapopulation Biology*, pp. 43–62. Academic Press, New York.