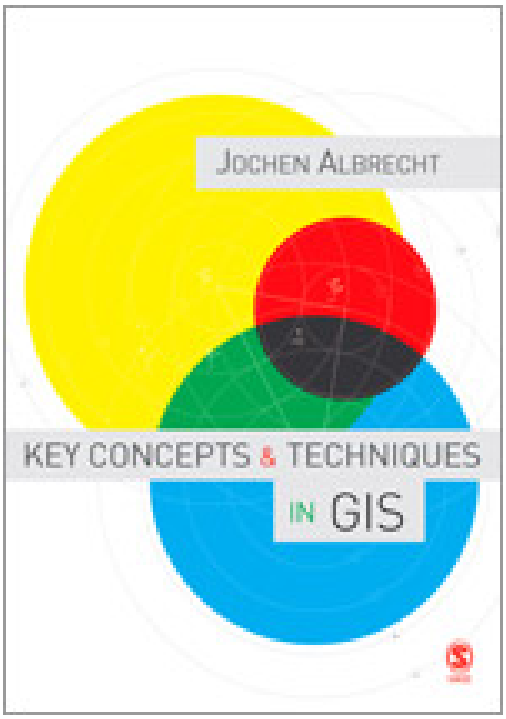


Sage Research Methods

Key Concepts and Techniques in GIS

For the most optimal reading experience we recommend using our website. A free-to-view version of this content is available by clicking on this link, which includes an easy-to-navigate-and-search-entry, and may also include videos, embedded datasets, downloadable datasets, interactive questions, audio content, and downloadable tables and resources.

Author: Jochen Albrecht

Pub. Date: 2011

Product: Sage Research Methods

DOI: https://doi.org/10.4135/9780857024442

Methods: Geographic information systems, Artificial neural networks, Databases

Keywords: map algebra, layer
Disciplines: Anthropology, Geography
Access Date: July 21, 2025
Publisher: SAGE Publications Ltd
City: London
Online ISBN: 9780857024442

Map Algebra

This chapter introduces the most powerful analytical toolset that we have in GIS. Map algebra is inherently raster-based and therefore not often taught in introductory GIS courses, except for applications in resource management. Traditional vector-based GIS basically knows the buffer and overlay operations we encountered in Chapter 6. The few systems that can handle network data then add the locationallocation functionality we encountered in Chapter 7. All of that pales in comparison to the possibilities provided by map algebra, and this chapter can really only give an introduction. Please check out the list of suggested readings at the end of this chapter.

Map Algebra was invented by a chap called Dana Tomlin as part of his PhD thesis. He published his thesis in 1990 under the very unfortunate title of *Cartographic Modeling* and both names are used synonymously. His book (Tomlin 1990) deserves all the accolades that it received, but the title is really misleading, as the techniques compiled in it have little if anything to do with cartography.

The term ‘map algebra’ is apt because it describes arithmetic on cells, groups of cells, or whole feature classes in form of equations. Every map algebra expression has the form <output = *function(input)*>. The function can be unary (applying to only one operand), binary (combining two operands as in the elementary arithmetic functions plus, minus, multiply and divide), or *n*-ary, that is applying to many operands at once.

We distinguish map algebra operations by their spatial scope; local functions operate on one cell at a time, neighborhood functions apply to cells in the immediate vicinity, zonal functions apply to all cells of the same value, and global functions apply to all cells of a layer/feature class. In spite of the scope, all map algebra functions work on a cell-by-cell basis. The scope only determines how many other cells the function takes into consideration, while calculating the output value for the cell it currently operates on (see Figure 37). However, before we get into the details of map algebra functions, we have to have a look at how raster GIS data is organized.

8.1 Raster GIS

Raster datasets can come in many disguises. Images raw, georeferenced, or even classified consist of raster

data. So do many thematic maps if they come from a natural resource environment, digital elevation models (see Chapter 9), and most dynamic models in GIS. As you may recall from Chapter 2, a raster dataset describes the location and characteristics of an area and their relative position in space. A single raster dataset typically describes a single theme such as land use or elevation.

Figure 37 The spatial scope of raster operations

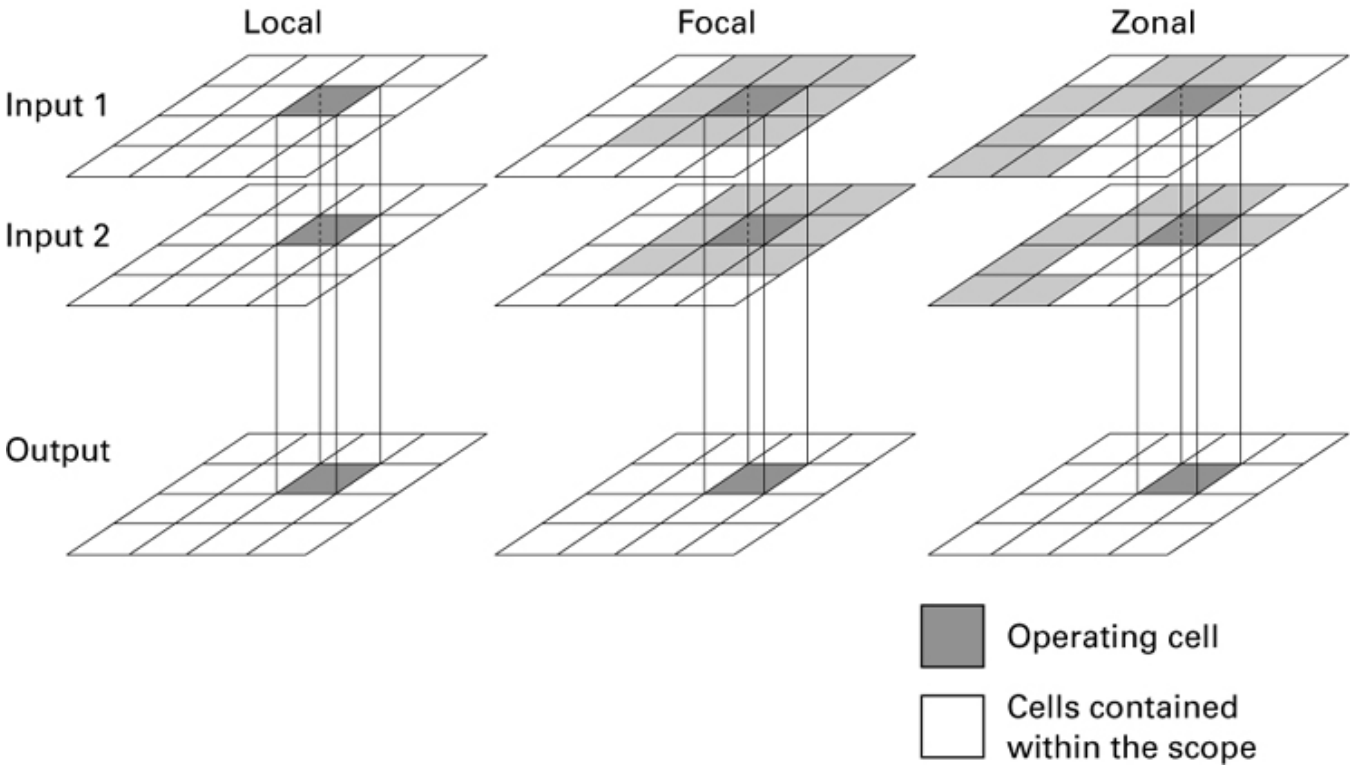
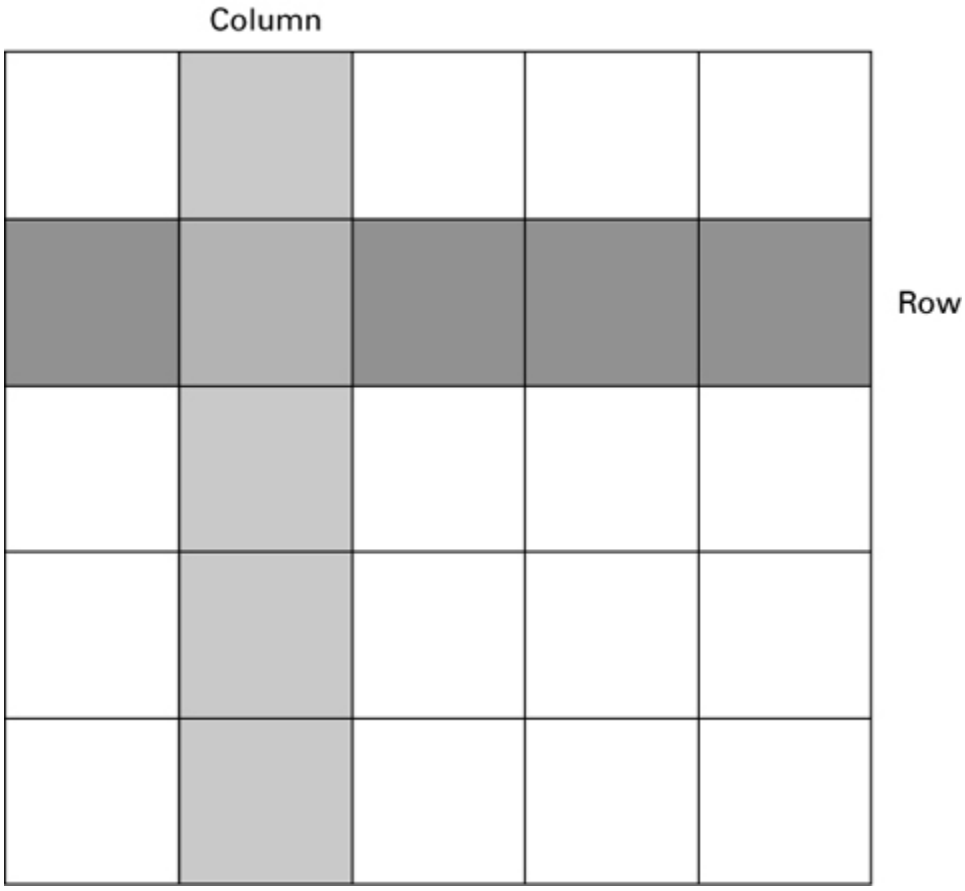
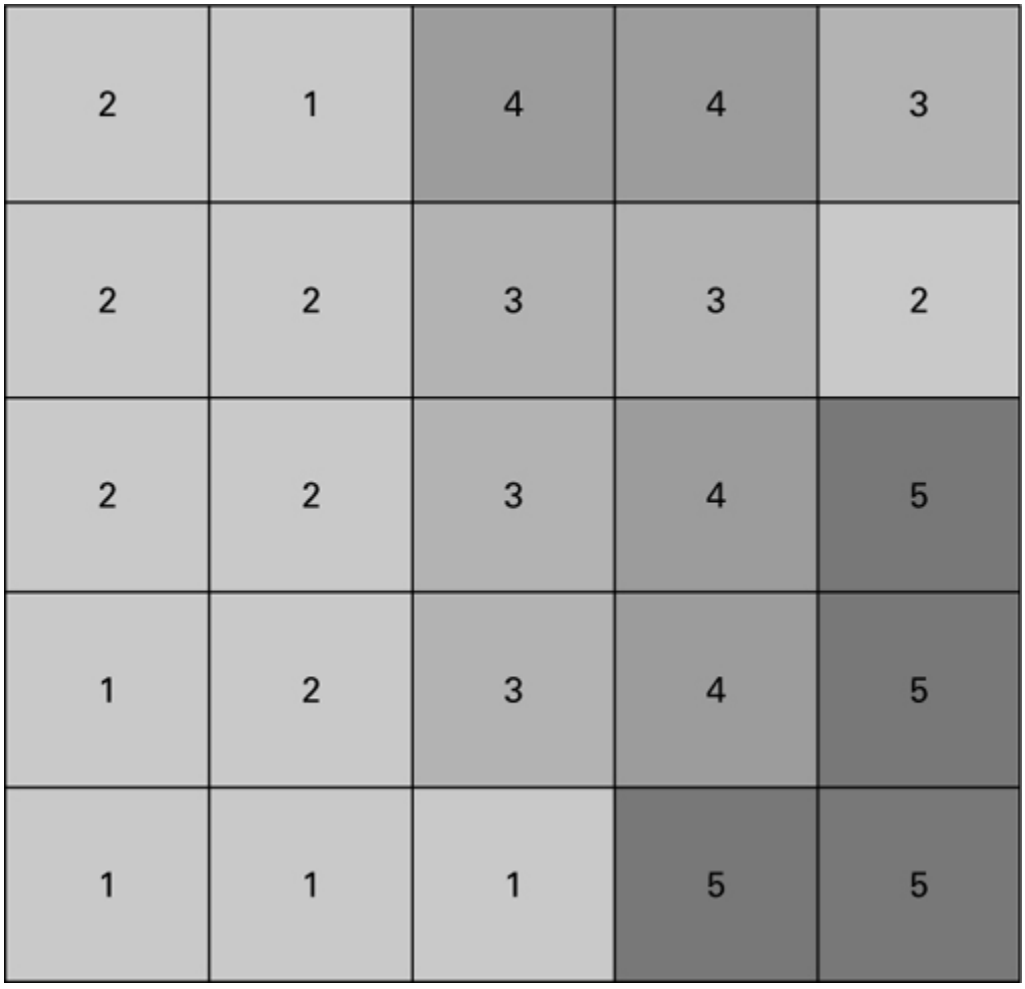


Figure 38 Raster organization and cell position addressing



At the core of the raster dataset is the cell. Cells are organized in rows and columns and have a cell value very much like spreadsheets (see Figure 38). To prove this point, Waldo Tobler, in a 1992 article, described building a GIS using Microsoft Excel; you are not encouraged to follow that example as the coding of GIS functionality is extremely cumbersome and definitely not efficient. Borrowing from the nomenclature of map algebra, all cells of the same value are said to belong to the same zone (see Figure 39). Cells that are empty that is, for which there is no known value are marked as NoData. NoData is different from 0 (zero) or 9999, or any other typically out-of-range value. Upon encountering a NoData cell, map algebra functions react in a well-defined way.

Figure 39 Zones of raster cells



Cell values can have two different purposes. They can represent a true quantitative value (e.g. elevation or amount of precipitation), or they can represent a class, whose values are then described in an external table. In the latter case, the cell value acts as a pointer to the correct record in the external table.

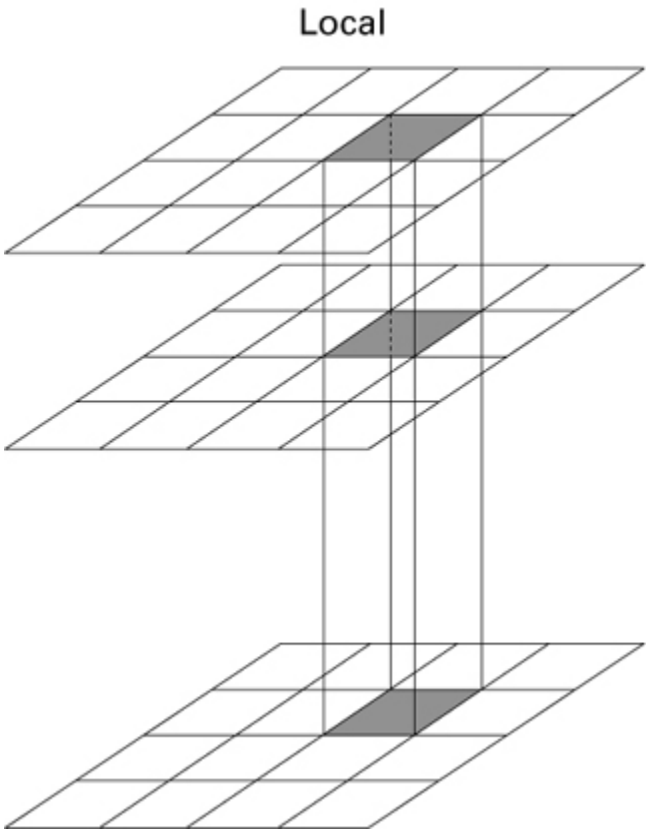
8.2 Local functions

All map algebra functions work one cell at a time. Local functions derive their name from the fact that in the calculation of the output value only input cells with exactly the same coordinate are considered (see Figure 40). The somewhat tedious description of the procedure goes as follows.

A local map algebra function reads the cell values of cell position (1,1) and applies a certain calculation on

these input values. It then writes the result to cell (1,1) in the output layer and proceeds to the second cell in the row, where the whole procedure is repeated, one cell at a time until we get to the last cell in the last row. It is easy to see that this would be very slow if each reading/writing step were to involve the hard disk. With the price of memory coming down, most GIS nowadays are able to process one or two layers (depending on their size) virtually before writing the result to a file or table. The cumbersomeness of the process is mitigated by the fact that no complicated geometric calculations have to be performed (as all cells are of the same size and orientation), and that the calculations themselves can often be executed within the processor itself, which makes it extremely fast. So, although a million cells may be processed, the result of a local operation is often instantaneous. Compare this to the complexity of overlay operations in the vector world (see Chapter 6)!

Figure 40 Local function



Local map algebra functions can be arithmetic, trigonometric, exponential, logarithmic, statistical or logical in nature. A trivial example would involve only one input layer, where we multiply all cell values by a constant value, say ‘3’. As a result, we could have an elevation layer, where the ratio of horizontal to vertical distances is now exaggerated enough to visually discern terrain features (see Figure 41).

One notch up on the ladder of sophistication is the use of a multiplier layer (see Figure 42). Say we have counties with different property tax rates. We could then calculate not just purchase prices but long-term costs by multiplying the costs of each property by the tax rates. Observe what happens to the cells with no values; NoData times something results in NoData.

Figure 41 Multiplication of a raster layer by a scalar

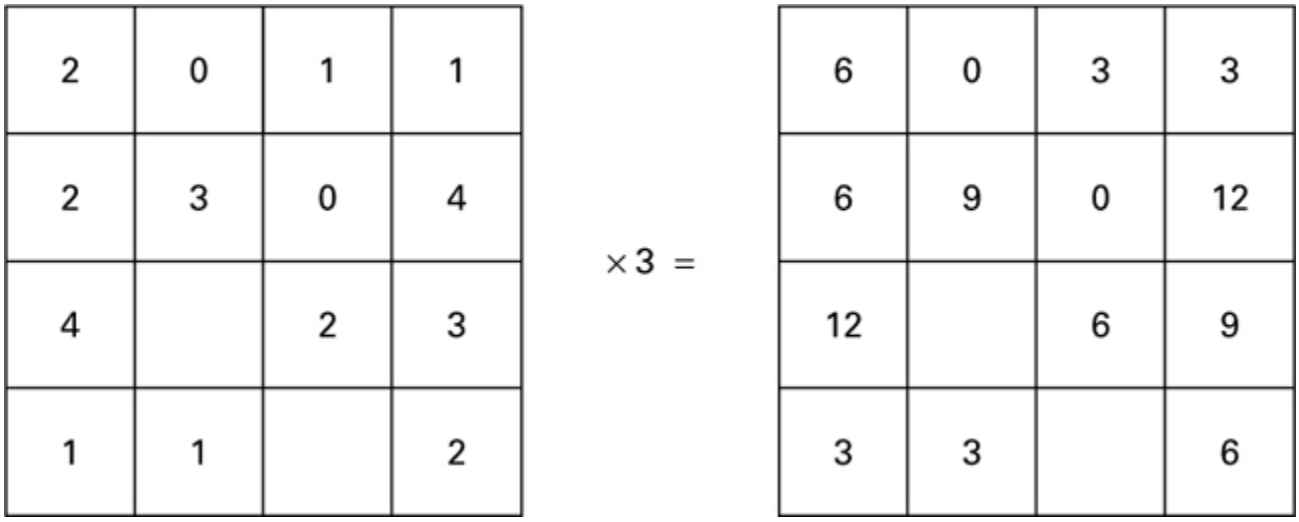
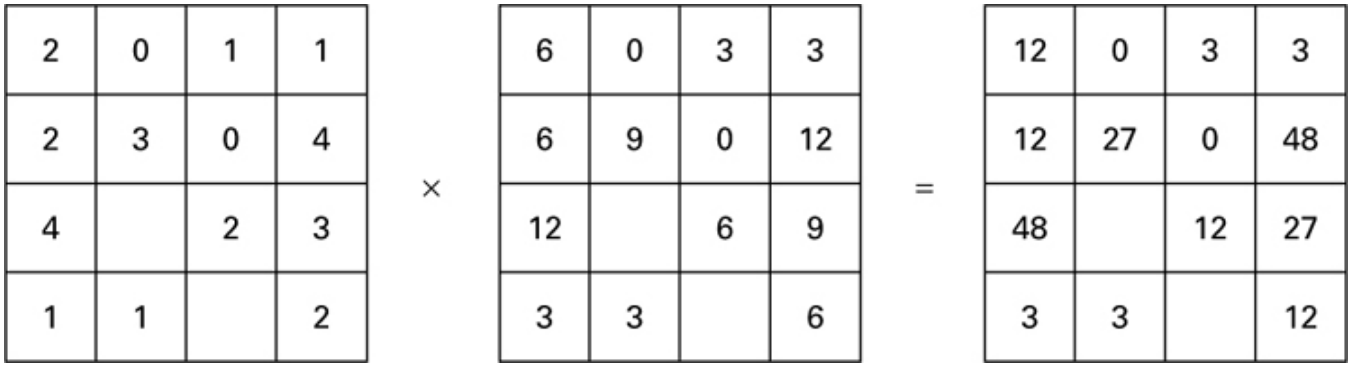


Figure 42 Multiplying one layer by another one

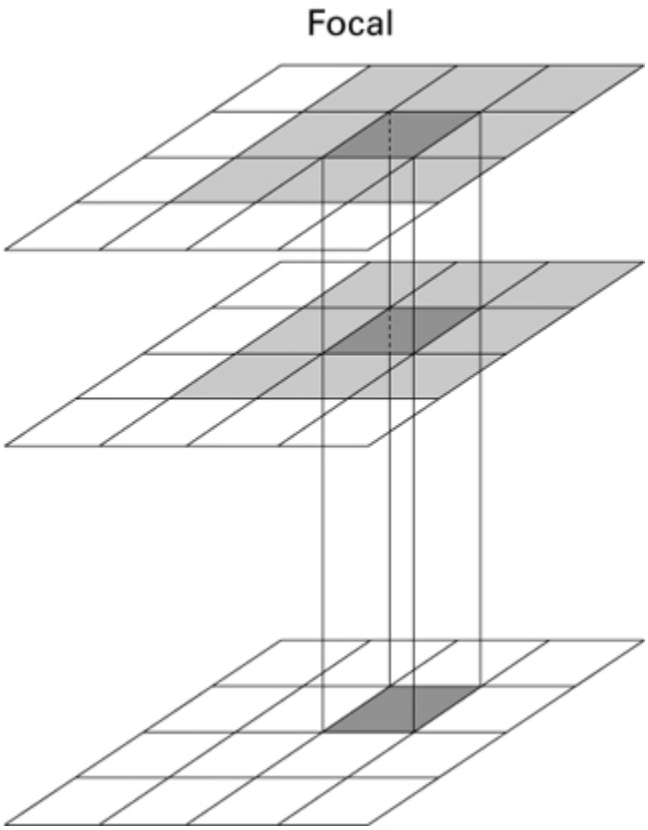


8.3 Focal functions

Neighborhood or **focal functions** are the main reason for the success of map algebra. Local functions, in a way, are nothing but fancy recoding operations, which could indeed easily be performed in a proper database management system. Focal functions, on the other hand, take into account all cells in a user-defined

neighborhood. Anybody who ever tried to calculate a cell value in a spreadsheet based on a combination of surrounding cells knows that the query string very quickly becomes really complicated. Again, the procedure is one cell at a time, except that for the calculation of an output cell value, we now look at all cells surrounding the processing location of all input layers (see Figure 43).

Figure 43 Focal function



By default, the neighborhood is defined to be all eight cells touching the processing cell, plus the processing cell in the middle. As an interesting aside, the first step of a focal function uses only three neighboring cells because cell (1,1) sits in a corner and has no more neighbors. Many GIS allow us to define different neighborhoods, though. We may change the extent (size) of the neighborhood, its shape, or its kernel (where in relationship to the neighborhood sits the operating cell?).

Just about any function type (arithmetic, trigonometric, exponential, logarithmic, statistical or logical) that we encountered for local functions is also applicable for focal functions. In addition, we have functions that compare cell values within a neighborhood, rather than across multiple input layers. This way, we can determine the average, minimum or maximum value within the neighborhood or calculate the span (range) of values.

One of the attractive features of map algebra is its naming conventions. Identifiers such as FocalSum, FocalMean, LocalAND or ZonalMax are more or less self-explanatory, yet this is exactly how all map algebra functions are formed. Actual vendor implementations change a little bit in their semantics (e.g. average versus mean) but stick to the principle.

Figure 44 describes a prominent focal function that is the basis for many image processing operations. You are encouraged to work the example, or at least the first row of cells in Figure 44. Remember to include the processing cell in all calculations. We will revisit neighborhood functions in Chapter 9, as the very notion of terrain is a focal one.

Figure 44 Averaging neighborhood function

2	0	1	1
2	3	0	4
4	2	2	3
1	1	3	2

=

1.8	1.3	1.5	1.5
2.2	2.0	1.8	1.8
2.2	2.0	2.2	2.3
2.0	2.2	2.2	2.5

8.4 Zonal functions

Zonal functions are in effect a mixture of local and focal functions. Based on Tobler's **First Law of Geography** (see Chapter 10), cells of similar values can be expected to lie next to each other. Hence everything that was said about focal functions applies to zonal functions as well. On the other hand, more important than spatial contiguity is the fact that all cells within a zone have the same value and they are treated the same. Zonal functions therefore take on the character of recoding functions that typified local functions.

The process is a bit more complicated than before. Again, zonal functions work on a cell-by-cell basis. However, the zone now acts as a kind of variable neighborhood definition (see Figure 45), and the zonal input

layer as a lookup table. A zonal function processes at least two input layers, one so-called zone layer, and one or more value layers. The zones can be (and usually are) but do not have to be contiguous.

Figure 45 Zonal function

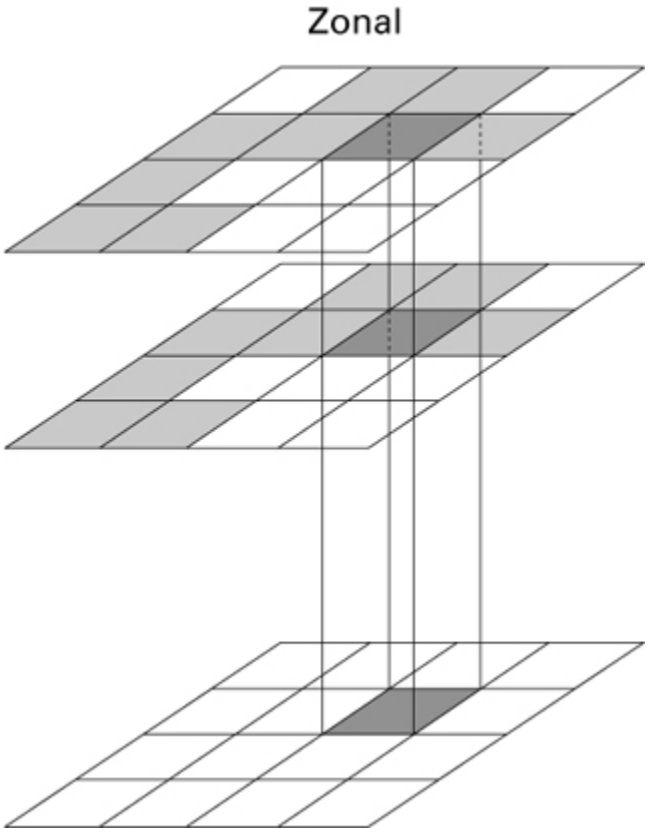
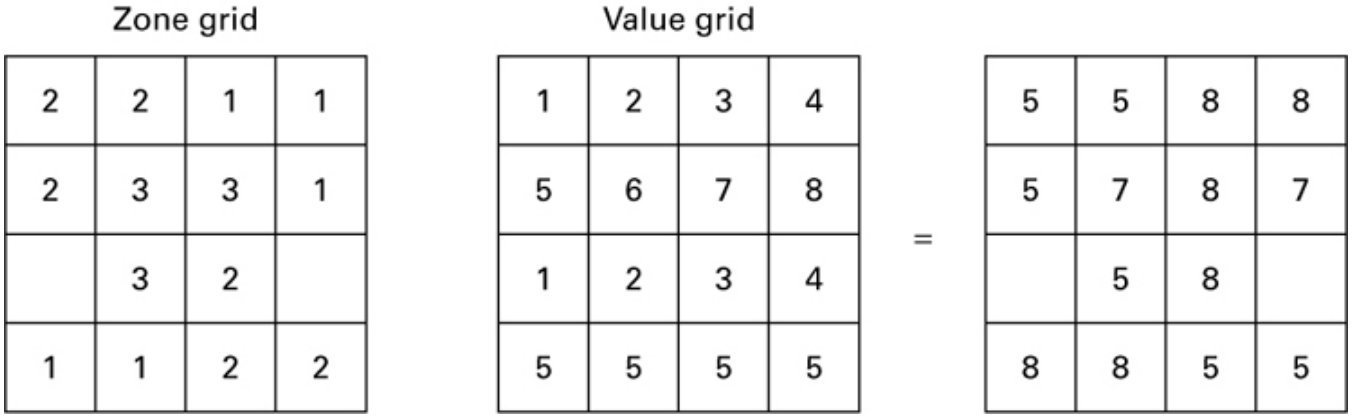


Figure 46 is an abstract example for a ZonalMax function. Again, as all map algebra functions, it starts at cell (1,1), and looks what zonal number it contains. Then it looks at all other cells with the same zonal number and creates a list of their values from the second input layer. Upon determining the maximum value, it writes that value to cell (1,1) in the output layer and proceeds to the next cell. If that next cell has the same zonal number then the output value from the previous step is written to cell (1,2). If the zonal number is different, then a new list of values belonging to all cells of the new zone number is created and its maximum is written to cell (1,2). In practice, the zonal number grids represent classified images or thematic maps, where we run calculations on the value layer based on the cell's membership to one class or another.

8.5 Global functions

Global functions differ from the rest in that many of the arithmetic or statistical operations do not make sense if applied globally (they would result in the same value for all cells of the output grid). Instead, global functions determine the spatial relationship between cells of interest or in other words, the distance between cells.

Figure 46 Value grids as spatial lookup tables



There are two types of global or distance function, Euclidean and weighted. The former is a straightforward geometric calculation, while the latter involves a second input layer that encodes a cost or friction surface. With global functions, it is fairly easy to implement the most complicated vector GIS operations: Thiessen polygons, corridors and locationallocation.

8.6 Map algebra scripts

One of the most convincing arguments for map algebra was Tomlin's use of map algebra scripts, a concatenation of functions that represents a complete GIS analysis workflow or even a dynamic model. While Tomlin just used simple sequences, authors like Kirby and Pazner (1990), Wesseling and van Deursen (1995) and Pullar (2003) developed temporal constructs that mimic classic programming languages augmented by spatial expressions. A simple example is the following script, which models the spread of a pollutant from a point source:

For *i* = 1 to 100

```
plume = buffer(plume)* 1/i
```

One of the interesting characteristics of most raster-based programs (particularly image processing packages) is that the target file can be overwritten in a programming loop. This saves disk space but of course prevents the analysis of intermediate steps. Scripts are also the mechanism behind the raster-based terrain modeling operations discussed in Chapter 9.

https://doi.org/10.4135/9780857024442