AN ABSTRACT OF THE THESIS OF

Linh Thu Nguyen for the degree of Master of Science in Geography presented on December 7, 1999. Title: A Spatial Model for Studying Population Dynamics of the California Mojave Desert Tortoise

Abstract Approved: ______________________________

Gordon Matzke

A spatially explicit population model written in C++ programming language is used in this study examining the population dynamics of the Desert Tortoise (*Gopherus agassizii*) in the California Mojave Desert. The model is constructed with hexagonal divisions of space, and with one-year increments of time. Four thematic maps: vegetation, soil type, roads, and elevation, are used to generate hexagon grids classifying the status of resources and roads in the modeled site. The grids are used as the input habitat data for runs of the model. Simulations used a projection matrix for seven size classes. The matrix contains the averages of demographic rates of tortoise population in different study sites in the Western Mojave (Doak et al. 1994). Four landscapes with a total area of 2736.21 km^2 are sampled from the study site. Main events taking place in one time step include movement, survival, and reproduction. The movement is modeled for eight size classes based on the mobility characteristics of the species and the size-specific mobilities. The number of animals moving out of a hexagon in every time step is based on the instantaneous resource gradient between that hexagon and its six neighbors. Reproduction and survival are responses to the site-specific status of roads and rainfall. Because of the serious lack of field data to support relationships hypothesized in the model, sensitivity analysis of model parameters was conducted. This study also examined the elasticity of elements of the projection matrix to see if the rankings of elasticities are different in a spatial context.

The results show that the road effect is the most sensitive parameter in the model. Differences in site-specific population dynamics can be well explained by local road status. Effects from spatial and temporal variation of rainfall do not show any obvious differences in population dynamics between sites. Simulations conducted at the finer scale show the dynamics of population better than at the coarser scale. Elasticities of vital rates in the projection matrix in the spatial model have the same ranking pattern as in a non-spatial model.

A Spatial Model for Studying Population Dynamics of the California Mojave Desert Tortoise

by

Linh Thu Nguyen

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirement for the
degree of

Master of Science

Presented December 7, 1999
Commencement June 2000

Master of Science thesis of Linh Thu Nguyen presented on December 7, 1999

APPROVED:

Major professor, representing Geography

Chair of Department of Geosciences

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Linh Thu Nguyen, Author

ACKNOWLEDGEMENTS

First, I would like to thank my major professor, Dr. Gordon Matzke, my advisor, Dr. Ross Kiester, and my minor professor, Dr. Jon Kimerling for their support and advice throughout my graduate study. During the course of conducting this thesis project, I want to give special thanks to Dr. Ross Kiester. Without his continuous guidance, encouragement, supervision and support, this study would not have been possible. I also wish to thank Dr. Martin Fisk for his willingness to serve on my committee.

Next, I would like to thank Kevin Sahr for his work on the C++ Library of Discrete Grid Systems, without which I could not imagine how I would get this project done. His work is appreciated, and his assistance is acknowledged.

My graduate study is fully funded by World Wildlife Fund (WWF) under the Vietnam GEF Project (VIE/91/G31). I would like to thank Mr. David Hulse, WWF Vietnam Country Representative, for his support in the past two years of my education at Oregon State University.

A number of individuals provided support in the completion of this project. People in Terra Cognita Lab, Mark Meyers, Lary Hughes, and Jill Heaton have been willing to offer me assistance whenever I needed it. I have greatly benefited from conversations with my friends, Duc Tran, Minh Le, Vinh Nguyen, and Tim Worley. I would like to thank all of them for both their support and friendliness.

I would like to thank my parents, my sister, Phuong Thuy, my brother-in-law, Viet Bac, and my niece, Thu Phuong for their enduring love, cares, and encouragement. It is their love that has encouraged me to finish this thesis.

Finally, my thanks go to Trung Nguyen for his love, understanding, and support. His patience has been the motivator that I have needed in order to complete this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

A Spatial Model for Studying Population Dynamics of the California Mojave Desert Tortoise

1. INTRODUCTION

Spatially explicit population models (SEPMs) are becoming increasingly useful tools for population ecologists, conservation biologists, and land managers. SEPMs are used to predict the responses of a population to land-cover change, climate change, and landscape fragmentation because these subjects are difficult to study with traditional ecological techniques (Levin, 1992). SEPMs allow one to change environmental variables in the model while holding life history characteristics of the modeled population constant, or vice versa (Dunning et al. 1995). Sensitivity analysis, which allows a variable of the model to vary while holding all the others variables constant, can be used to determine the relative importance of different variable values on the model's results. SEPMs also are used as the guidance for field data collection needed to increase the accuracy of the model (Dunning at al. 1995), and to save time and funding spent on field work to provide unneccesary data while critical data are left uncollected. SEPMs can be a useful tool to examine the consequences of different proposed land-use scenarios, and suggest the best scenario satisfying desired purposes (Schumaker 1995).

In this research, a spatially explicit demographic simulator is constructed to study the dynamics of the California Mojave desert tortoise (*Gopherus agassizii*) population. Doak et al. (1994) developed a non-spatial demographic model for the species in the western Mojave Desert. It is stated in their paper that "In the absence of frequent dispersal between subpopulations and without independent increase and decrease in local densities, extinction/colonization processes or rescue effects can, at best, be of only secondary importance in determining population dynamics. In this situation there is little point in pursuing spatial modeling". This conclusion is the very motivator for my research to study a spatial model for the tortoises. Most of the demographic data used in their model were collected over a time interval of less than ten years, and some data were not collected annually in that period. This short time interval cannot represent the effect of the spatial and temporal variation of climate factors in the desert, and a non-spatial model can not include the movement behavior of the animals. All these elements may affect population dynamics in the long-term. Additionally, tortoises are long-lived species. Adult tortoises may be more than 40 years old. Individuals require about a decade or longer to reach sexual maturity (Bury 1982). Therefore, to sustain the tortoise populations, long-term management plans are urgently needed.

The desert tortoise is a herbivorous reptile inhabiting the California Mojave Desert. The animals stay in their under-ground burrows to avoid the hardship of the winter and summer weather on the Desert. Therefore, vegetation that provides food and soil types that provide burrow-digging environments are important limiting factors for the species. Tortoise populations reportedly have undergone large declines during the past century (Bury and Corn 1995, Luckenbach 1982). Threats to the species led the U.S. Fish and Wildlife Service (USFWS) to give an emergency endangered-species listing in 1989, and list the species as threatened in 1990 (Doak et al. 1994, Bury and Corn 1995).

Berry (1984) and U.S. Fish and Wildlife Service (1994) concluded that the most convincing factors explaining the declines in the western Mojave Desert are human activities including: collection for pets and food, vandalism, mortality on roads, military training activities, off-road vehicles, urbanization, and conversion of desert to crop lands. Humans can also have less blatant detrimental impacts that operate by modifying the tortoises' relationship with other species. Three such indirect threats to tortoises are thought to be especially important: livestock grazing, raven predation, and upper respiratory tract disease (URTD) (Doak et al. 1994). Even though those factors may have serious impacts on tortoise populations, the lack of quantitative studies on these matters prevents me from addressing those threats in this project. A lot of research and effort have been expended to save the species. In this study, a model is developed to examine the potential long-term population dynamics and to identify the most critical demographic and environmental factors relevant to the species' survival.

Sahr (1998) has constructed a C++ library of Discrete Grid Systems. This library is used to build the spatially explicit population simulator in this project. The model is hexagon-based and size-based, which means it keeps track of the information about habitat and number of individuals in each size class in each hexagon through time. The population statistics are calculated after each one-year time step.

As mentioned above, the main factors causing the decline in the population are human activities that mostly target individual animals. Therefore, the natural dynamics of habitat are less important for the fate of the species, and the spatial model assumes a static habitat over time.

Initially, the elements used to construct the model and test the sensitivity of each element with respect to the total population are examined. Second, the elasticity of each entry in the projection matrix in both spatial and non-spatial models is analyzed (Caswell 1989). Because of the serious lack of field data used to construct the model, sensitivity analysis is a useful method to provide guidance for future data collection for studying the population as well as for improving the uncertainty of the model itself.

Elasticity of a static projection matrix, which is non-spatial, is commonly used as guidance for managerial work (Doak et al. 1994, Mills et al. 1998). Therefore, studying elasticity of the matrix in a spatial model is worth considering to see if it acts differently when spatial variations are included. For all these reasons, the main purpose of this model is not to quantitatively predict the dynamic of the tortoise population in reality, but to focus on a relative ranking of the potential effects of the most critically natural and human-related factors and demographic factors on the dynamics of the tortoise populations. This allows for approaching a more realistic model for the tortoise by constructing and improving a spatial simulator for the species.

2. METHODS

2.1. Habitat Data

Luckenbach (1982) and Schamberger and Turner (1986) have concluded that most critical habitat factors for tortoises in the California Mojave Desert are vegetation, soil type, elevation, and annual rainfall. In this study, vegetation, soil, and elevation data themes are combined to produce the input resource hexagon map for the model. The effects of annual rainfall and road status on the tortoise population are included and discussed in the "Survival and Reproduction Modeling" section of this thesis.

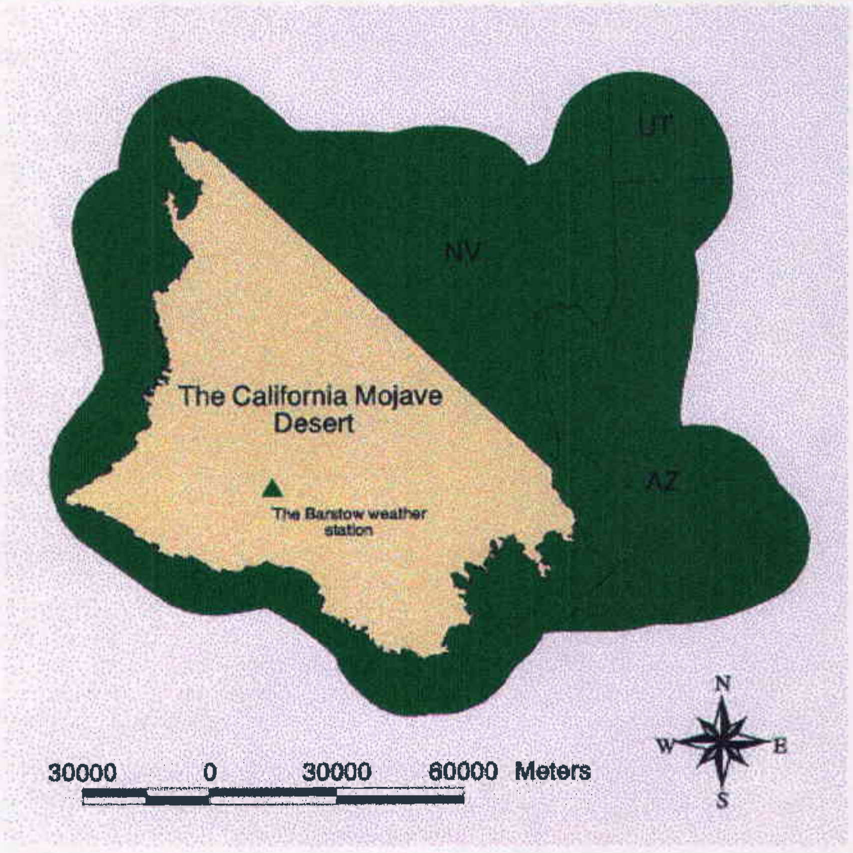

UT
NV
AZ
The California Mojave Desert
The Barstow weather station
30000 0 30000 60000 Meters
N
W
E
S

Figure 1. The study area, California Mojave Desert, in the entire Mojave Desert Ecoregion.

Vegetation provides food and water for animals. Soil properties affect the easiness of their burrow digging process. Luckenbach (1982) found that in the California Mojave Desert, the desert tortoise mostly frequents four plant communities: creosote scrub, cactus scrub, shadscale scrub, and Joshua tree woodland; occasionally tortoises occur in alkali scrub. In terms of soil type preference, the study notes that probably no type is preferred, but the type must be friable enough for the digging of burrows and firm enough so that burrows will not collapse. And it is also found in this study that the tortoises rarely occur above 1,000 m throughout the western Mojave Desert. All these pieces of information about tortoise habitat preferences are translated quantitatively in the form of a Habitat Suitability Index in the study by Schamberger and Turner (1986). (Figures 2 and 3)

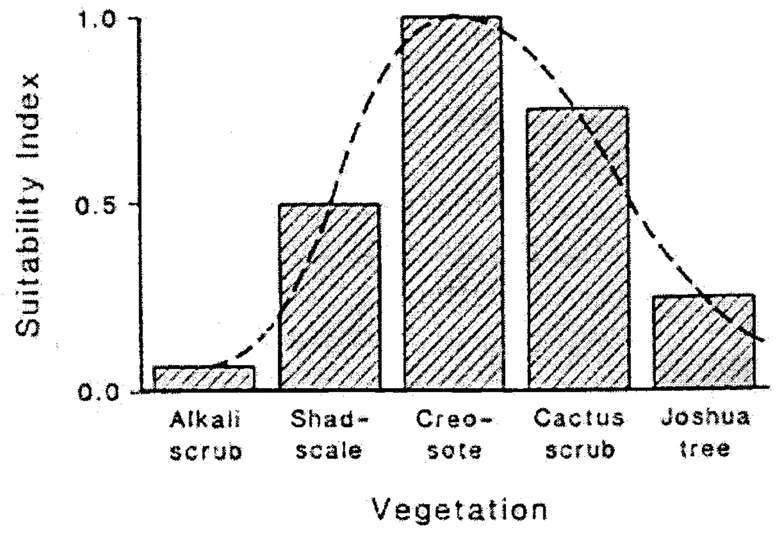


Figure 2. Vegetation and Habitat Suitability Index for the Desert Tortoise
(from Schamberger and Turner (1986))

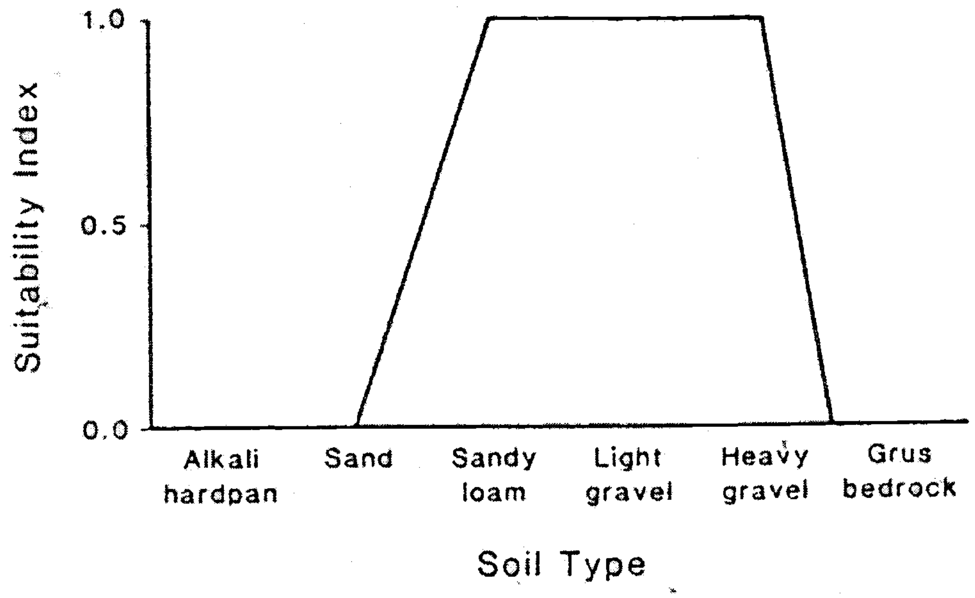


Figure 3. Soil Type and Habitat Suitability Index for the Desert Tortoise
(from Schamberger and Turner (1986))

In this study, I used three vector coverages (vegetation, soil type, and road), and one raster layer (digital elevation model (DEM) of entire Mojave ecoregion) compiled by the Mojave Desert Ecosystem Program (Appendix A). For the road coverage, roads were divided into 4 main categories: highway, pave road, dirt road, and off-road. Each road type was assigned a weight that reflected the potential effect of roads on the tortoise population. This road coverage was rasterized to get a road grid. The value of each pixel in the grid corresponded to an assigned weight in the vector layer. The higher the weight, the more dangerous the road type with regards to tortoise population effects.

Information from the two graphs above is used to translate the vegetation layer and the soil type layer into the new grids, which have the value of the corresponding Habitat Suitability Index (HSI). These two new grids are called HSI Vegetation grid, and HSI Soil Type grid. The resolution of the road grid, vegetation grid, and soil type grid is

3-arc-seconds. Therefore, the DEM is resampled to a grid of resolution 3-arc-seconds to get consistency in resolution among the three raster layers used for further manipulation. The DEM is translated into a bit map, with the value of 0 representing the elevation above 1000 m, where no tortoises occur, and value of 1 representing the elevation equal or less than 1000 m, where the presence of the animal is possible. To create the GIS layer as the input habitat for my spatial model, I use Arc/Info to generate a new grid, whose value represents the geometric mean of the HSI Vegetation grid and HSI Soil Type grids. The formula of geometric means is

$$\text{geometric mean} = \text{sqrt (HSI Vegetation * HSI Soil Type)} \qquad \text{(Equa. 1)}$$

The geometric mean is a typical way to combine effects caused by two different factors. This new grid is further multiplied by the bitmap elevation grid to mask out the region of elevation above 1000m, which means that no tortoises are found in this area. This final grid is called the Combined HSI grid. Four landscape subsets, each with an area of 2736.21 square kilometers, are generated from this grid, and will be used as the inputs for the model simulations later on. These four subsets are chosen to satisfy the wide range of site-specific habitat quality and the status of resource availability and roads (see Figures 4-7).

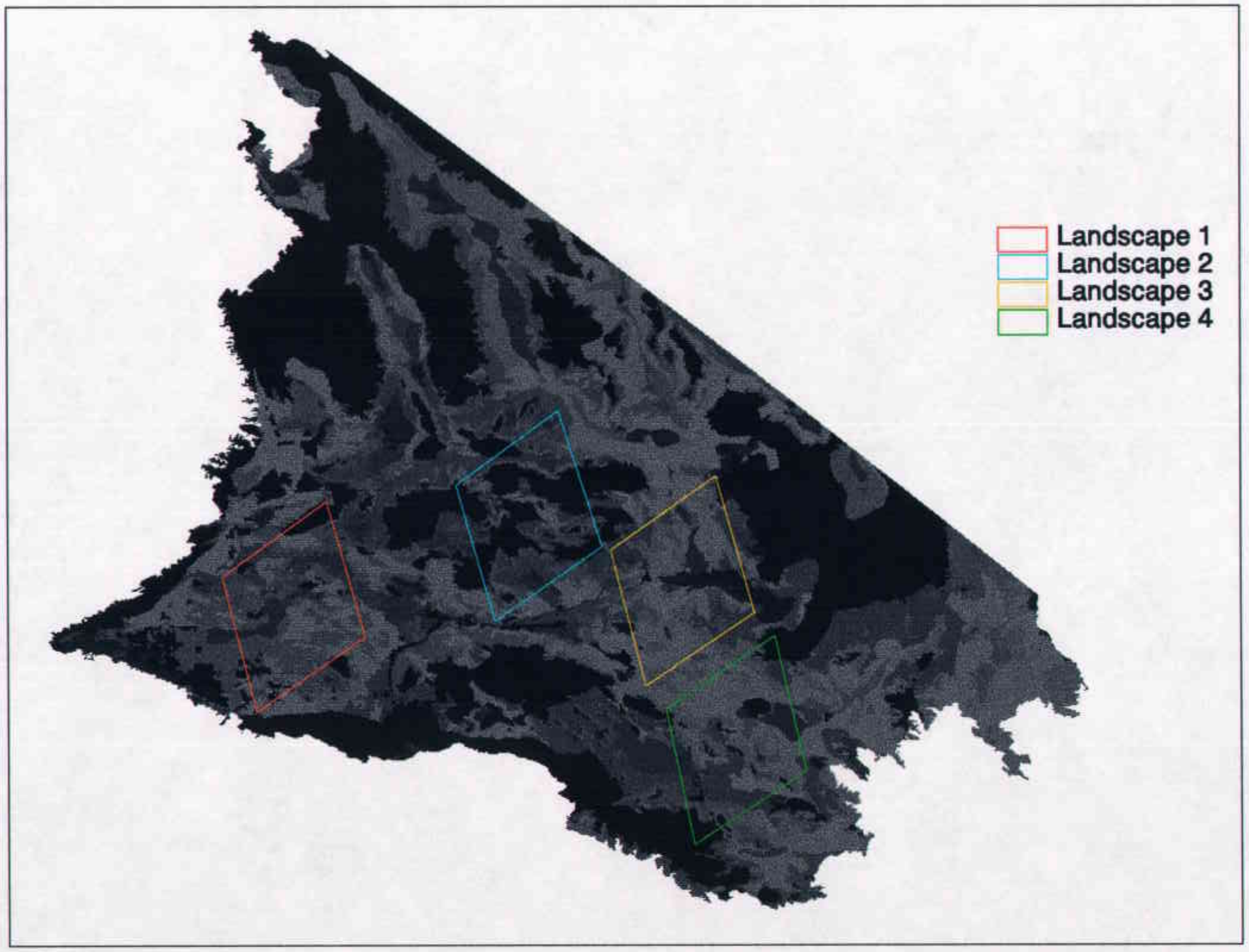


Figure 4. Locations of four sample landscapes in the resource grid

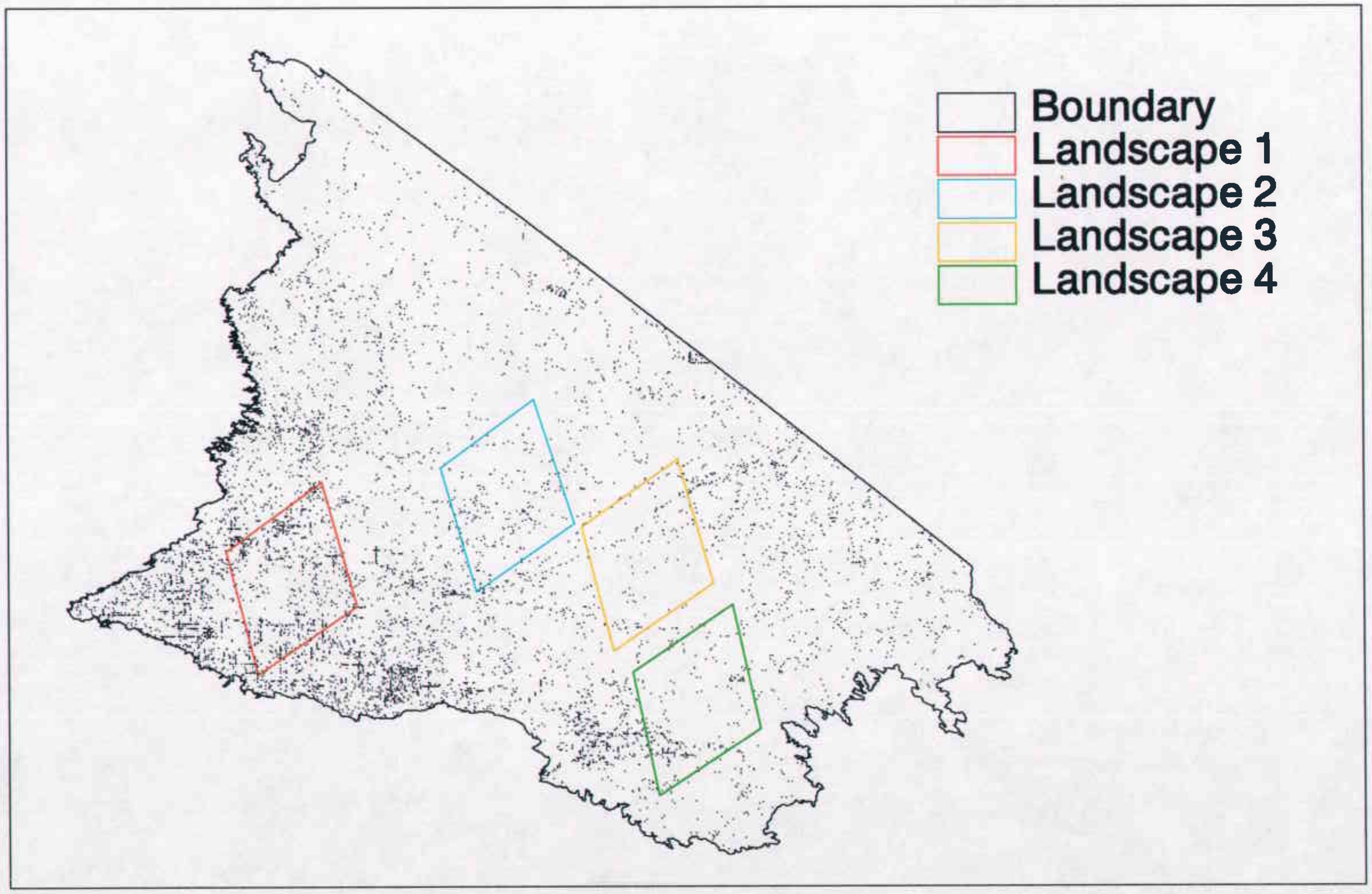


Figure 5. Locations of four sample landscapes in the road-value grid

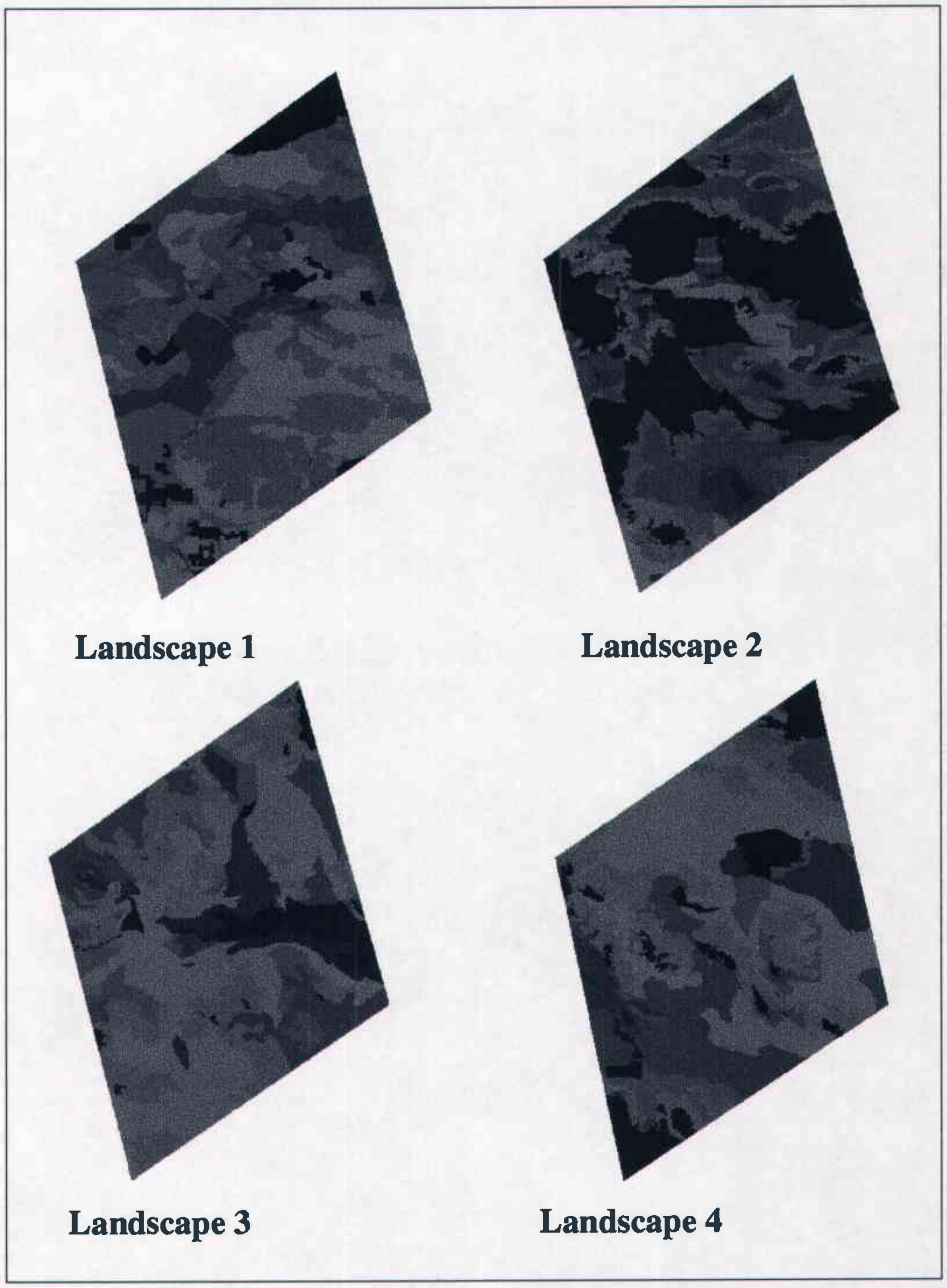


Figure 6. Resource grids in four sample landscapes

The brighter the color, the higher the resouce values. The range of the resource values is from 0 to 100. The resource value of each pixel is calculated based on its Habitat Suitability Index (HSI) combined from HSIs of vegetation, soil, and elevation.

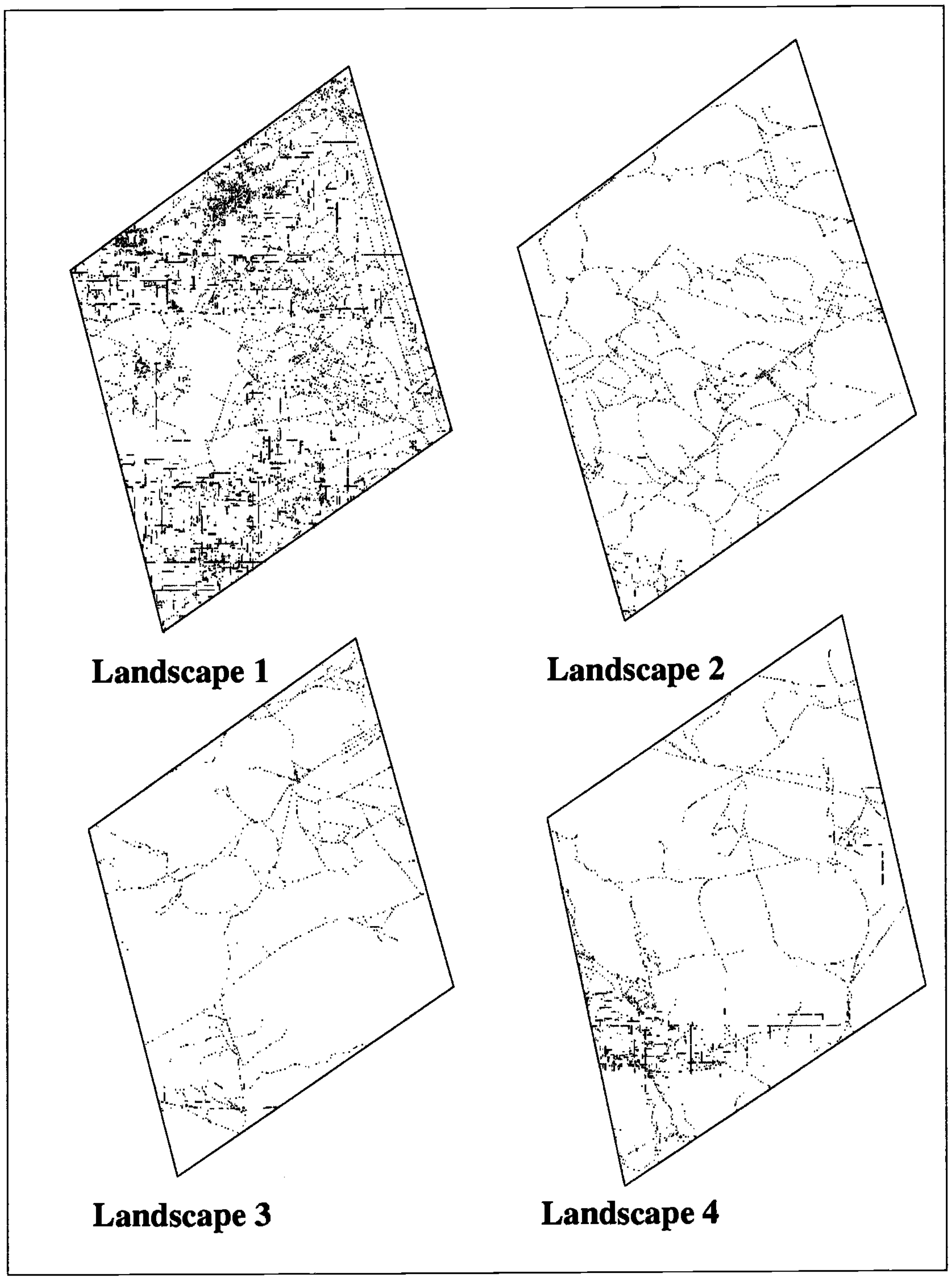


Figure 7. Grids of road values in four sample landscapes

A C++ program, called the pixel-binning program, is used in this process to intersect the Combined HSI grid and the road grid with a regular array of hexagons covering the study area. The output of this program assigns the hexagon number to each pixel in the input grid. The Combined HSI grids and road grids for the four sample landscapes are binned to hexagon grids of resolution 14 and 16 on the Snyder projection (Kimerling et al. 1994). The area of each hexagon is 10.68832 sq.km and 1.18759 sq.km at resolution 14 and 16, respectively (Figure 8). The output files of the Combined HSI grids are then used to calculate the habitat score for each hexagon, which is the weighted average of the HSI values of all pixels falling in that hexagon multiplied by 100. Since the range of the original HSI values is from 0 to 1, the HSI of each hexagon varies from 0 to 100.

The output files from the road grids are used to calculate the road value for each hexagon, i.e. the sum of road values of all pixels contained in the hexagon divided by the total number of pixels in the hexagon. The road values of all hexagons then are rescaled to have a range from 0 to 100.

2.2. Model Structure

I wrote the spatial model in C++ programming language by using common C++ standard libraries and a library of Discrete Grid Systems (DGGSs). I used layers of habitat data thought to be critical to the tortoises to generate a new GIS layer that specifies the location and quality of each hexagon in the area of interest. This layer is used as the habitat or resource input for the model. Before a simulation is run, each hexagon has the information of habitat quality and a certain initial number of animals.

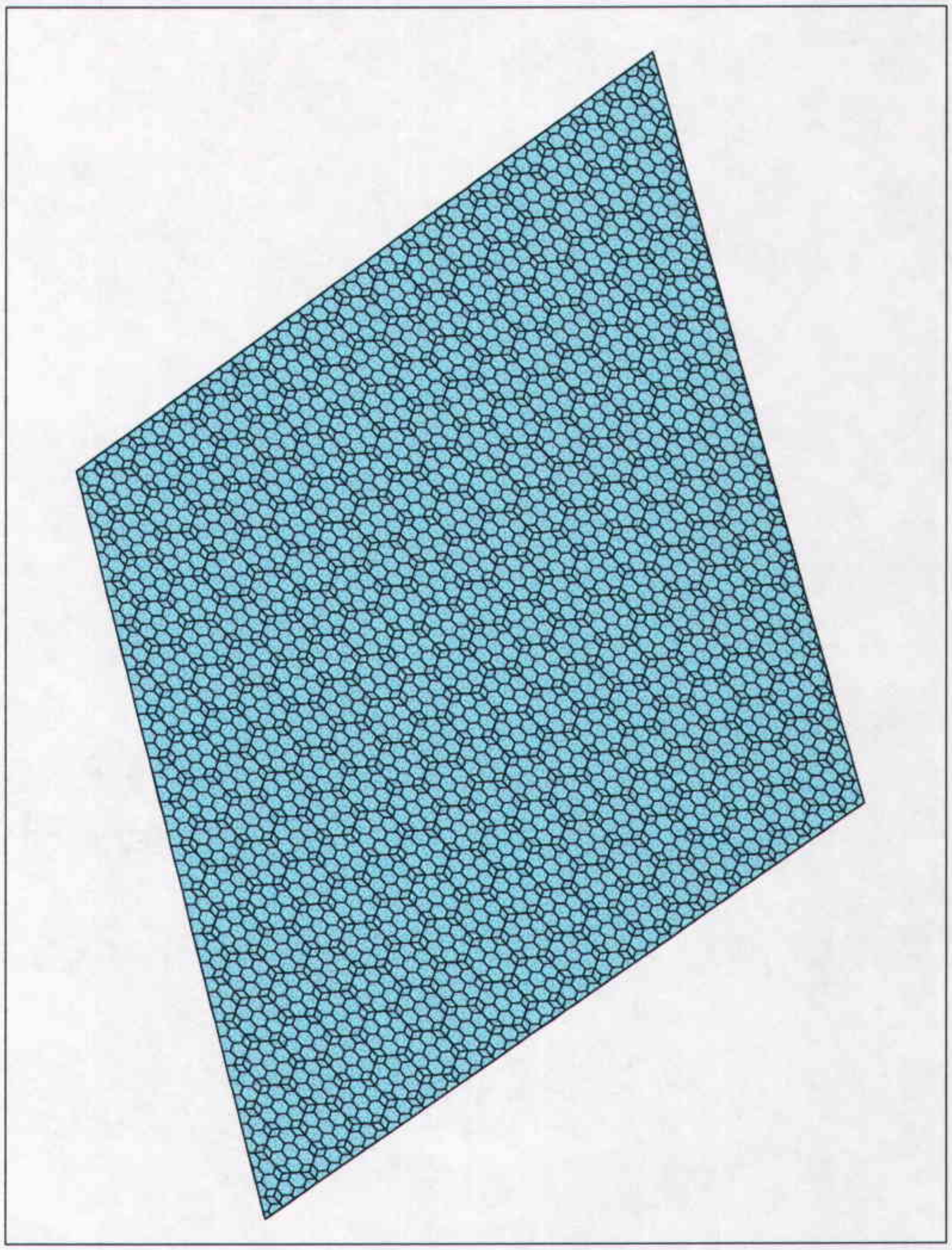


Figure 8. Overlay of grid resolution 14 and grid resolution 16 in a sample landscape
A hexagon has an area of 1.18759 sq.km at resolution 16 and 10.68832 sq.km at resolution 14. The total area of each sample landscape is 2736.21 sq.km.

Time steps for the model are one year, during which both movement and reproduction of the tortoise population occur once. These activities are modeled on a hexagon basis. The movement algorithm controls the density of animals in each hexagon with respect to the amount of available resource in that hexagon and its six neighboring hexagons. Survival and reproduction are modeled in each hexagon based on a fixed projection matrix, the spatial and temporal variation in rainfall, and the status of roads in the area. After each time step, the total number of animals in the population and the growth rate of the population are calculated and saved.

2.2.1. Movement modeling

The movement function used in the model is based on the movement pattern presented by Kiester and Slatkin (1974) and Kiester (1985). The idea is that animals respond to the difference between the amount of resource available in the site and neighborhood they are occupying. I borrowed the idea of these two original papers, in which the models are simulated in a one dimensional array of patches, and applied it to two dimensional landscapes, i.e. hexagon grids, and to females only. It is assumed that the tortoises in the host hexagon (Hhex) only move to one of six neighboring hexagons, which is called the best neighboring hexagon (BNhex), and that the tortoises cannot move further than a distance of one hexagon in a time step, i.e. a year. In the case that no neighboring hexagon is better than the situation in the host hexagon, the tortoises will stay in their site and no movement occurs. If movement happens, the total number of moving animals from the host hexagon to the best neighboring hexagon is calculated by equation (2):

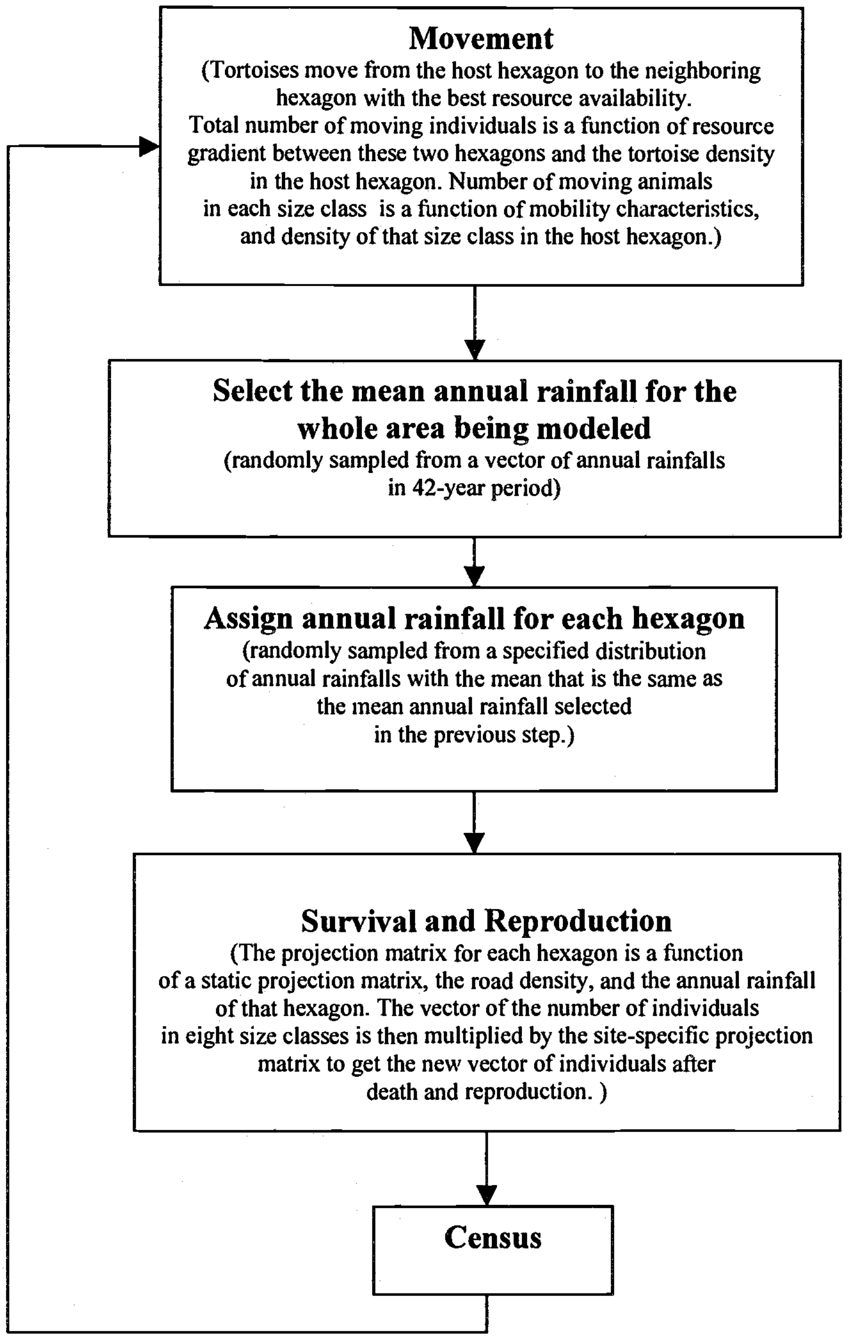

Movement
(Tortoises move from the host hexagon to the neighboring hexagon with the best resource availability.
Total number of moving individuals is a function of resource gradient between these two hexagons and the tortoise density in the host hexagon. Number of moving animals in each size class is a function of mobility characteristics, and density of that size class in the host hexagon.)
Select the mean annual rainfall for the whole area being modeled
(randomly sampled from a vector of annual rainfalls in 42-year period)
Assign annual rainfall for each hexagon
(randomly sampled from a specified distribution of annual rainfalls with the mean that is the same as the mean annual rainfall selected in the previous step.)
Survival and Reproduction
(The projection matrix for each hexagon is a function of a static projection matrix, the road density, and the annual rainfall of that hexagon. The vector of the number of individuals in eight size classes is then multiplied by the site-specific projection matrix to get the new vector of individuals after death and reproduction.)
Census

Figure 9. The flow of modeled events for a single year

$$NMoving = a * n(Hhex) * (n(Hhex)/res(Hhex) - n(BNHex) / res(BNHex))$$

(Equa. 2)

where:

a : a constant

n(Hhex): number of individuals in the host hexagon

res(Hhex): number of resource units in the host hexagon

n(BNHex): number of individuals in the best neighboring hexagon

res(BNHex): number of resource units in the best neighboring hexagon

This equation is called the Moving Equation. In the equation, the total number of moving animals is a function of the tortoise density in the host hexagon and the gradient between resource availability in the host hexagon and in its best neighboring hexagon. The best neighboring hexagon is the neighbor that has the highest gradient of resource pressure, i.e. lowest number of individuals per unit resource, compared to the host hexagon.

From now until the end of this section, it is assumed that there is occurrence of movement. The total number of moving animals out of the host hexagon is calculated from the above equation. The next issue is the size-specific number of moving individuals. In this model, the population is divided into eight size classes, named size 0 to size 7. The reason for this division is that in my model, I use the projection matrix that is constructed for eight size classes of female tortoises from Doak et al.(1994) (Tables 1-2).The matrix is averaged from different sampling sites in their study in the western Mojave Desert. Size-structured demographic models are used instead of age-structured models because the task of aging tortoise individuals is very difficult, and the largest and most comprehensive data base for tortoise growth and mortality is indexed by size rather than age (Doak et al.1994, Berry 1984).

Coming back to the issue of movement, the question is: How many animals in each size class contribute to the total number of moving animals? Each size class is associated with a moving weight, and the sum of all moving weights of eight size classes is one. Hence the number of moving animals in each size class is the product of its moving weight and the total number of moving animals calculated in the Moving Equation. The moving weight of each size class is calculated by the following equation:

$$mw(\text{size } x) = mobility(\text{size } x) * n(\text{size } x)/sum(mobility(\text{size } i) * n(\text{size } i))$$

(Equa. 3)

where

mw(size x) - moving weight of size x

n(size x) and n(size i)- number of individuals of size x and size i in the host hexagon, and i is integers from 0 to 7.

so that the number of moving individuals in each size class depends on the mobility characteristics of that size class, and the number of individuals of that size class in the host hexagon. The more mobile the size class and/or the more crowded the size class in the host hexagon, the higher the number of individuals in that size class that will move. The mobility characteristics of all size classes are specified in a vector, and in the form of relative mobility between the eight size classes. Since there are no field data about the mobility characteristics of different size classes to support the model, the content of this vector will vary in a few different ways, and the effects of that variation on the tortoise population dynamics will be studied.

2.2.2. Reproduction and survival modeling

The next step in the model is to simulate the reproduction and survival of the population. The model uses matrix population modeling (Caswell, 1989), but it is modeled in a spatially explicit context. Population dynamics are represented by a projection matrix, which contains information about size-specific fecundity, survival, probability of staying in the same size class, and probability of moving to the next size class after a time step. The basic idea of the matrix population modeling is that the vector of size-specific population for the next time step is projected by multiplying that vector in the current time step with the projection matrix. In the model, I use a projection matrix from Doak et al. (1994). In their study, four different projection matrices were constructed by averaging demographic parameters of tortoises in different study sites in the Western Mojave Desert. The four matrices have the same probabilities of surviving and growing from one size class into the next largest one. The only difference among them is the estimates of yearling reproduction. In this model, the most optimistic estimate of reproduction is used, which corresponds to a projection matrix with the total population growth rate every year equal to 0.982 (see Table 1).

With respect to the method for modeling the effects of rainfall, the question of whether or not annual rainfall can represent the rain effect on the tortoise population arises. What happens if two years have the same annual rainfall, but in one year rain occurs evenly throughout the year, while in the other rain is concentrated in a few big storms and drought conditions prevail throughout the remainder of year.

	Size class							
Size class	0	1	2	3	4	5	6	7
0	0	0	0	0	0	2.22	3.38	4.38
1	0.716	0.567	0	0	0	0	0	0
2	0	0.149	0.567	0	0	0	0	0
3	0	0	0.149	0.604	0	0	0	0
4	0	0	0	0.235	0.56	0	0	0
5	0	0	0	0	0.225	0.678	0	0
6	0	0	0	0	0	0.249	0.851	0
7	0	0	0	0	0	0	0.016	0.86

Table 1. Average population projection matrix for the Desert Tortoise in Western Mojave.
This matrix contains three kinds of elements: production rates of size 5, 6, and 7 (the non-zero elements in the first row); the probabilities of surviving and growing from one size class to the next largest one (the non-zero off-diagonal elements); and the probabilities of surviving but remaining the same size (the non-zero diagonal elements). (from Doak et al., 1994)

Class	Name	Maximum carapace length (mm)
0	Yearling	
1	Juvenile 1	< 60
2	Juvenile 2	60 - 99
3	Immature 1	100 - 139
4	Immature 2	140 - 179
5	Subadult	180 - 207
6	Adult 1	208 - 239
7	Adult 2	> 240

Table 2. Desert tortoise size classes used in the model (From Doak et al. 1994)

To answer this question, I conducted the analysis of correlation between the actual annual rainfall and the number of events per year, i.e. the number of days in the year rain occurred, from the data recorded in Barstow weather station in the period from 1939 to 1980. The results (Figure 10) show a relatively high correlation between those two factors ($r^2 = 0.65$, $F_{1,40} = 74.8$, $P < 0.001$)

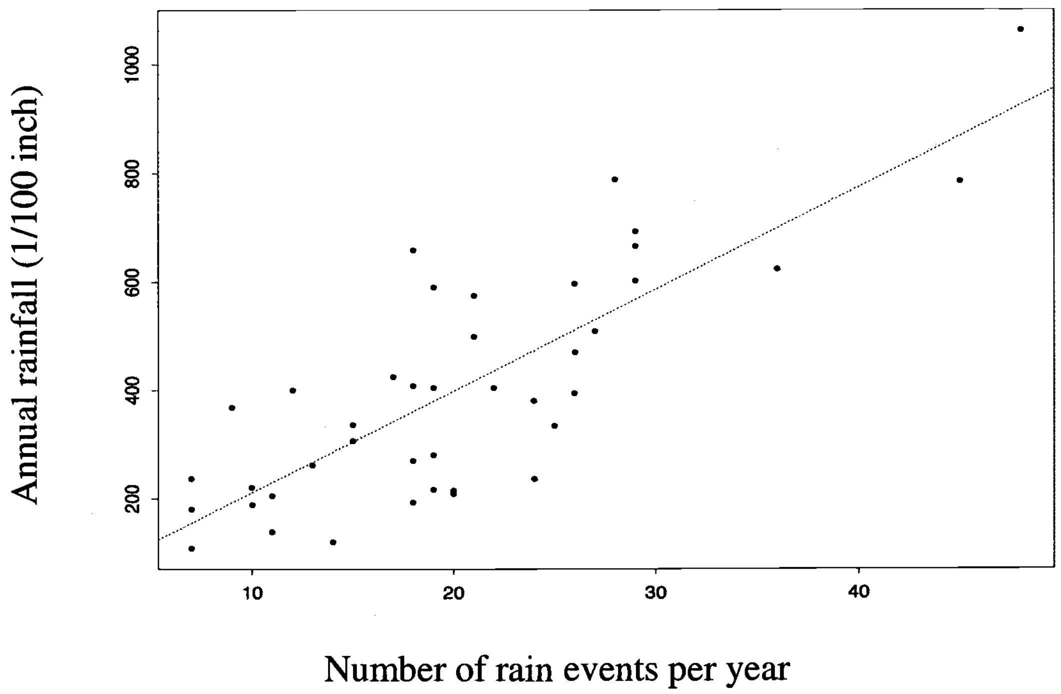


Figure 10. Correlation between annual rainfall and number of rain events per year in California Mojave Desert. Data are recorded from Barstow weather station in 42 year period (1939-1980). The line is the regression fit.

The temporal variation of rainfall within a year and between years is presented in the contour plot of monthly rainfall in that 42 year period (Figure 11). The figure shows that within a year, most of rain occurs in two main seasons, which are winter rain (from November to April), and scattered summer rain (from July to September). From the contour plot and the correlation results, I find that annual rainfall can be used to model the rain effect on tortoise population dynamics with relatively high reliability.

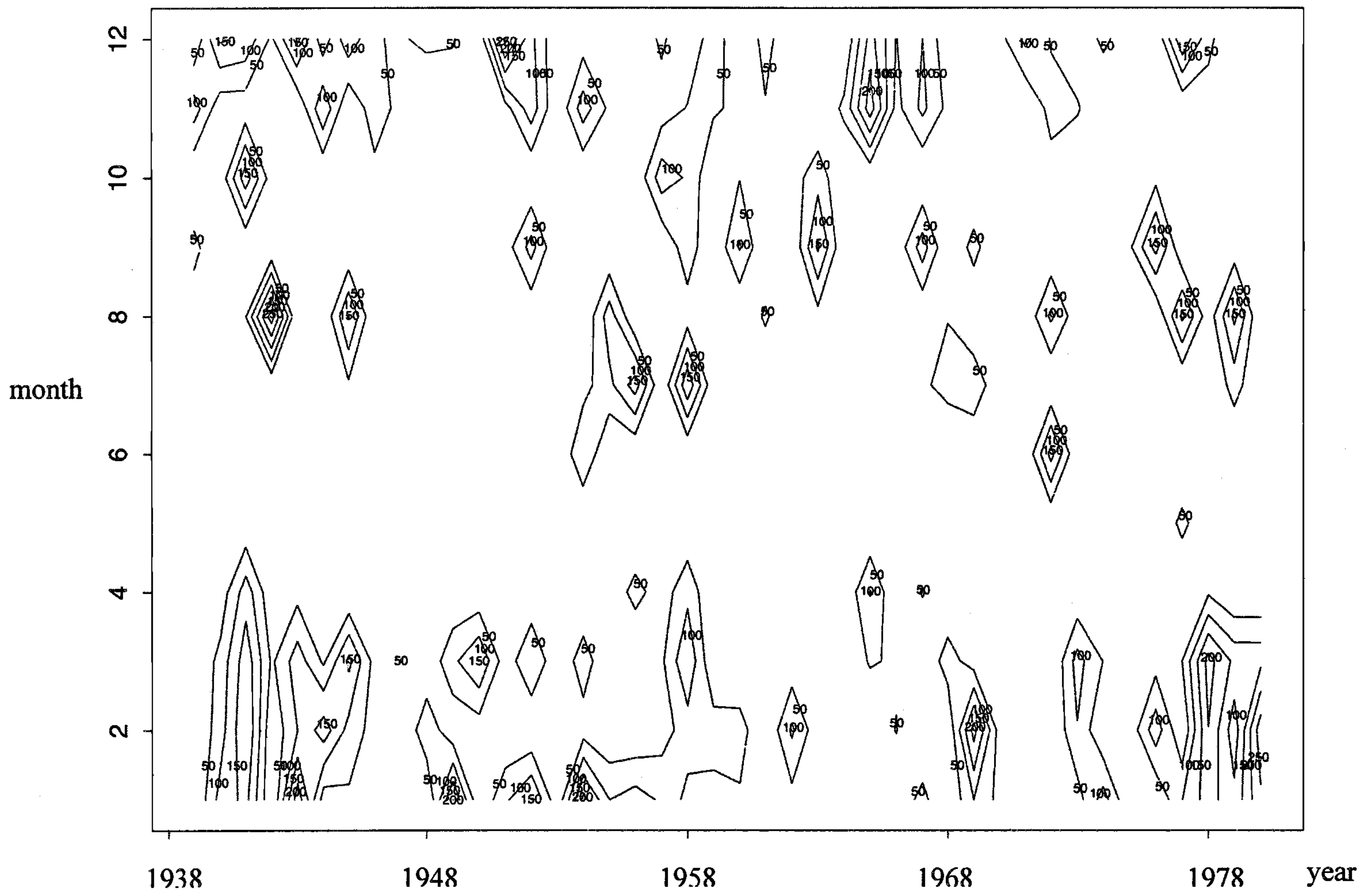


Figure 11. Contour plot of monthly rainfall in 42-year period (1939-1980) in the California Mojave Desert. Data are recorded from Barstow station. Web site: Southern California Climate summaries (http://www.wrcc.dri.edu/cgi-bin/cliMAIN.pl?cabars+sca)

To make it easier to follow, I call the projection matrix *the static projection matrix*. I assume that the matrix is the projection matrix corresponding to the means of all human-related and environmental factors included in my model, i.e. the mean of road density and the mean of annual rainfall.

It is further assumed that the variations in those factors have linear relationships with, and equal effects on, all elements of the static projection matrix. The effect by roads, effect by rain, and the combined effect of the two are represented in the model by variables named roadFactor, rainFactor, and combinedFactor, respectively. In each time step, the roadFactor and rainFactor variables are calculated, and then multiplied together to get the value for the combinedFactor. The combinedFactor will be further multiplied with the static projection matrix to get the site-specific projection matrix for that time step. The model checks the condition so that

```
if (combinedFactor > 1.07)    combinedFactor = 1.07
```
(Equa. 4)

This equation guarantees that all elements in the projection matrix vary in a reasonable range, which means the survival rates of all size classes are less than 1. The number 1.07 is chosen because when the combinedFactor is higher than 1.07, the reasonable variations of the matrix's elements are no longer satisfied. This checking routine, when applicable, increases the uncertainty of the model. However, by running simulations many times with different parameters, I found that the occurrences of the routine were rare.

The equations to calculate the roadFactor and rainFactor for each hexagon in a time step are as follows:

```
roadFactor = 1 - (a * (road value - MEAN_ROAD) / MEAN_ROAD)
```
(Equa. 5)

```
rainFactor = 1 + (b * (rainfall - MEAN_RAIN) / MEAN_RAIN)
```
(Equa. 6)

where

- **a, b** are constants

- **road value** is the mean of road value per pixel in the hexagon. Road values are re-scaled to range from 0 to 100. The road value for each hexagon is stable through time. Road values of all hexagons in the landscape represent the spatial variation of human impact on tortoise population by building roads.

- **MEAN_ROAD** is the mean of road values for all hexagons in the four landscapes.

- **rainfall** for each hexagon is randomly sampled from a vector of 30 elements that are calculated based on the mean annual rainfall for the whole landscape in the modeled time step. The content of this vector will be discussed in detail in Appendix B. This vector represents the spatial variation of rainfalls in the landscape. The mean annual rainfall is randomly sampled from the vector of 42 annual rainfalls recorded at the Barstow Weather Station from 1939 to 1980 (see Figure 16). This vector of annual rainfall represents the temporal variation of rain in the landscape.

- **MEAN_RAIN** is the mean of annual rainfall in the 42-year period

As mentioned above, MEAN_RAIN and MEAN_ROAD are assumed to be the **road value** and the **rainfall** corresponding to the projection matrix that is the same as the static projection matrix.

From equation (5), we can see that if the road value of a hexagon is equal to MEAN_ROAD, the roadFactor will equal one. The higher the road value, the smaller the roadFactor, which indicates a more negative effect on tortoise population. For example, if a site has a lot of roads built, it will be a bad sign for tortoise population growth, and the tortoise density in the site must be low.

From the equation (6), we can see that if rainfall of a hexagon is equal to MEAN_RAIN, the rainFactor will equal one. The higher the rainfall, the higher the rainFactor or the larger the values of elements in the projection matrix, which indicates a more positive effect on tortoise population. For example, if a site has a lot of rain in a time step, it will be a good sign for tortoise population growth in that year.

To control the infinity of population growth in hexagons in good conditions in terms of resource, road, and rain, the model checks the condition that:

If (resourcePressure > 6) combinedFactor = 1/0.982 (Equa. 7)

where 0.982 is the dominant eigenvalue or the population growth rate lambda of the static projection matrix. I use number 6 here to control the limitation of resource pressure, since I have never seen resource pressure higher than this number in any published papers. The meaning of equation (7) is that if the site-specific population reaches the level that resource pressure is larger than 6, the population in that hexagon will not grow in the instant time step, which means the population growth rate lambda (λ) equals one. The population in that hexagon can grow again if the individuals move or die.

2.2.3. Census

After each time step, the total population, annual rainfall, and population growth rate lambda will be calculated and saved to an output file. In this study, the term population growth rate lambda is simply the result from dividing the population of the hexagon in the current time step by the population in the previous time step. Additionally, the cell-specific information also can be saved if desired. The cell-specific information includes the hexagon ID, the rainfall, the road value, and the population of the site every year.

2.3. Sensitivity Analysis

Because of the serious lack of necessary data to support many quantitative relationships set up in the model, I arbitrarily generated some different ways to draw the relationships based on nature of each type of relationship and available published papers, and then conducted the sensitivity testing to roughly estimate the importance of the specific value of each parameter to the model results. Sensitivity analysis is able to aid the researcher in understanding which components should be measured most carefully. In this study, sensitivity analysis is conducted by making a series of runs while varying model parameters over expected or possible ranges, and recording the results for further analysis.

2.4. Elasticity of Projection Matrix Elements

Firstly, some related definitions in matrix population modeling need to be clarified. A vector x, with the property that matrix multiplication is equivalent to scalar multiplication:

$$Ax = \lambda x \qquad \text{(Equa. 8)}$$

for some scalar λ, is called an eigenvector of matrix A; the scalar λ is called the eigenvalue (Caswell 1989). The eigenvectors defined above are the right eigenvectors of A. A vector y is called left eigenvector of A if

$$y'A = \lambda y' \qquad \text{(Equa. 9)}$$

where y' is the transpose of y, and x, y, and λ can be real or complex.

Leslie (1945) transforms this idea of matrix algebra into matrix population modeling, where A is the population projection matrix. The topic is further developed by

Lefkovitch (1965), and Caswell (1989). It is proved that if A is primitive, the long-term dynamics of the population are described by the population growth rate λ_1 and the stable population structure w_1 (Caswell 1989). In the Tortoise case, w_1 is the stable size distribution of eight size classes, which corresponds to the dominant eigenvalue λ_1 of the projection matrix.

In the matrix model, the sensitivity of λ to changes in element a_{ij} of the projection matrix is expressed in the equation:

$$\partial\lambda/\partial a_{ij} = \upsilon_i \varpi_j / \langle w,v \rangle \qquad \text{(Equa. 10)}$$

That is, the sensitivity of λ to changes in a_{ij} is proportional to the product of the ith element of the left dominant eigenvector (v) and the jth element of the right dominant eigenvector or stable stage distribution (w). $\langle w,v \rangle$ is the scalar product of w and v. The elasticity, e_{ij}, of λ to element a_{ij} is simply the sensitivity rescaled to account for the magnitude of both λ and a_{ij}.

$$e_{ij} = (a_{ij}/\lambda)(\partial\lambda/\partial a_{ij}) \qquad \text{(Equa. 11)}$$

Thus, e_{ij} gives the proportional change in λ resulting from a proportional change in a_{ij}, while other elements remain constant. The growth rate λ is the dominant eigenvalue of the projection matrix (λ_1) (Caswell 1989). In this study, elasticities of demographic rates of both the static projection matrix and the spatial simulator are calculated and examined. The purpose of elasticity analysis is to answer the question whether or not the ranking of elasticities is unaffected by spatial factors. The elasticities of the static projection matrix are calculated by using a C++ template matrix library. Elasticities of the spatial model are calculated by varying each matrix element with a certain proportion and running model simulations. The results from elasticity analysis in a non-spatial and a spatial model are compared.

3. RESULTS

3.1. Results of Hexagon-Based Habitat Maps

Resource and road data are binned to hexagon grids of resolution 14 and 16 for four sample landscapes (Figures 12- 15). Landscape 1 has the highest road values relative to the other landscapes. Landscape 2 is the most fragmented, since a lot of its areas falls into the region marked out by elevation higher than 1000 m. Landscapes 3 and 4 are in rather good condition, with homogeneous characteristics and rich resources. However, the patterns of resources and roads in these two landscapes show a difference. Therefore, landscape 3 and 4 are chosen to test the sensitivity of model to the spatial distribution of resources and roads, which may affect the movement pattern.

3.2. Population Dynamics in Four Sample Landscapes

The study by Luckenbach (1982) concluded that the density of Desert tortoises in the Western Mojave vary within the range from 0 to 400 per sq.km. Since the range of resource values within a site is from 0 to 100, the initial number of animals in each hexagon was set equal to the resource pressure of 4. I have noticed that this initial number of animals should be divided by 2, since this spatial model is for females only. But then by examining the model structure carefully, I concluded that the scale of the initial number of animals was not important for the model's results. Within a site, the distribution of size classes were set to follow the stable size distribution of the static projection matrix, i.e. its right dominant eigenvector.

Population and population growth rate in the four sample landscapes are projected for a 100-year period using the habitat maps described above. Simulations are run with different parameter combinations to study the possible tortoise population trends in

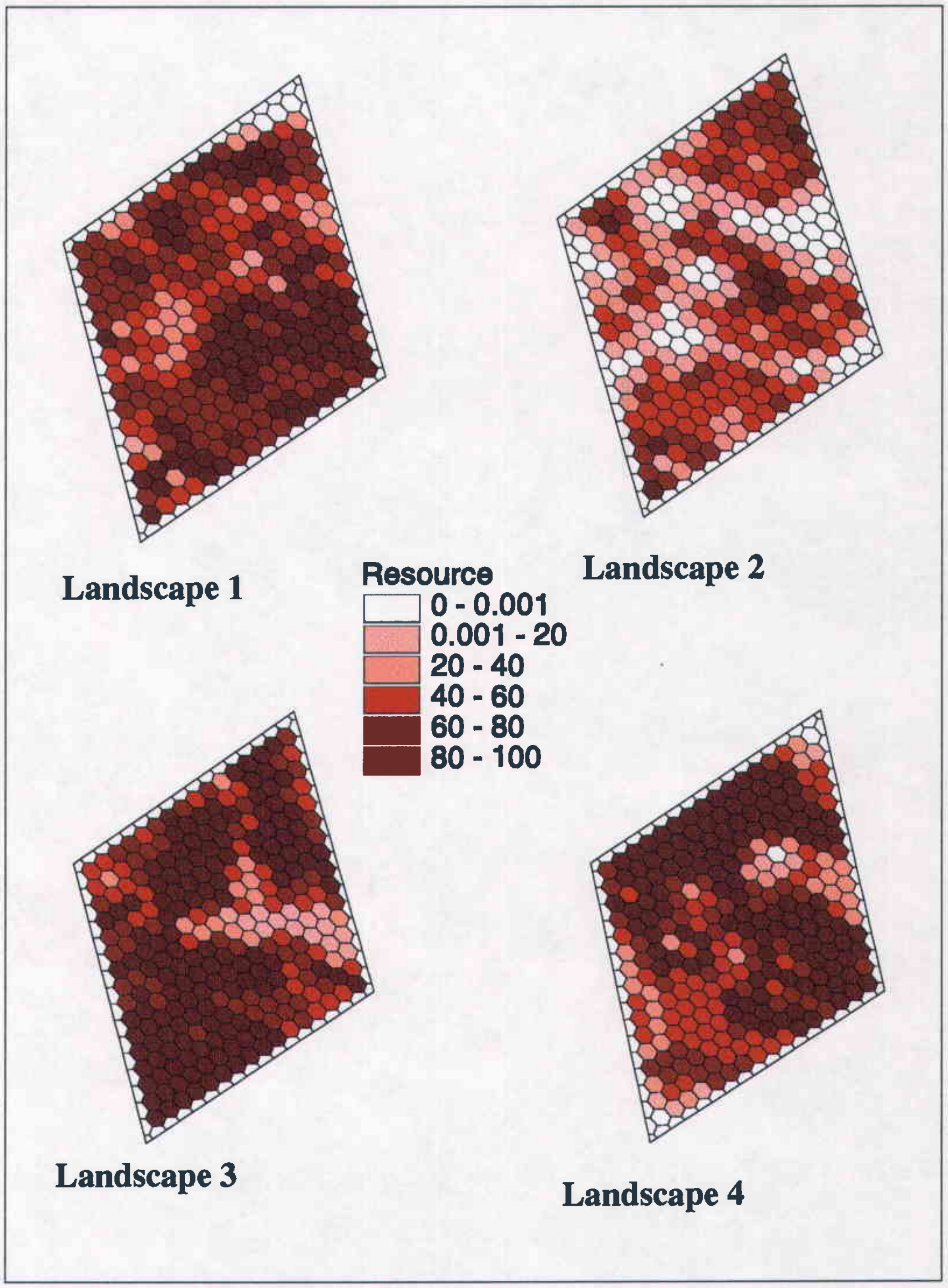


Figure 12. Hexagon map of resource in four sample landscapes (resolution 14)
The resource value of each hexagon is the average of resource values of all pixels belonging to it.The resource value of each pixel is calculated from Habitat Suitability Indices (HSI) of vegetation, soil, and elevation.
The higher the resource values, the more suitable the habitat for tortoises.

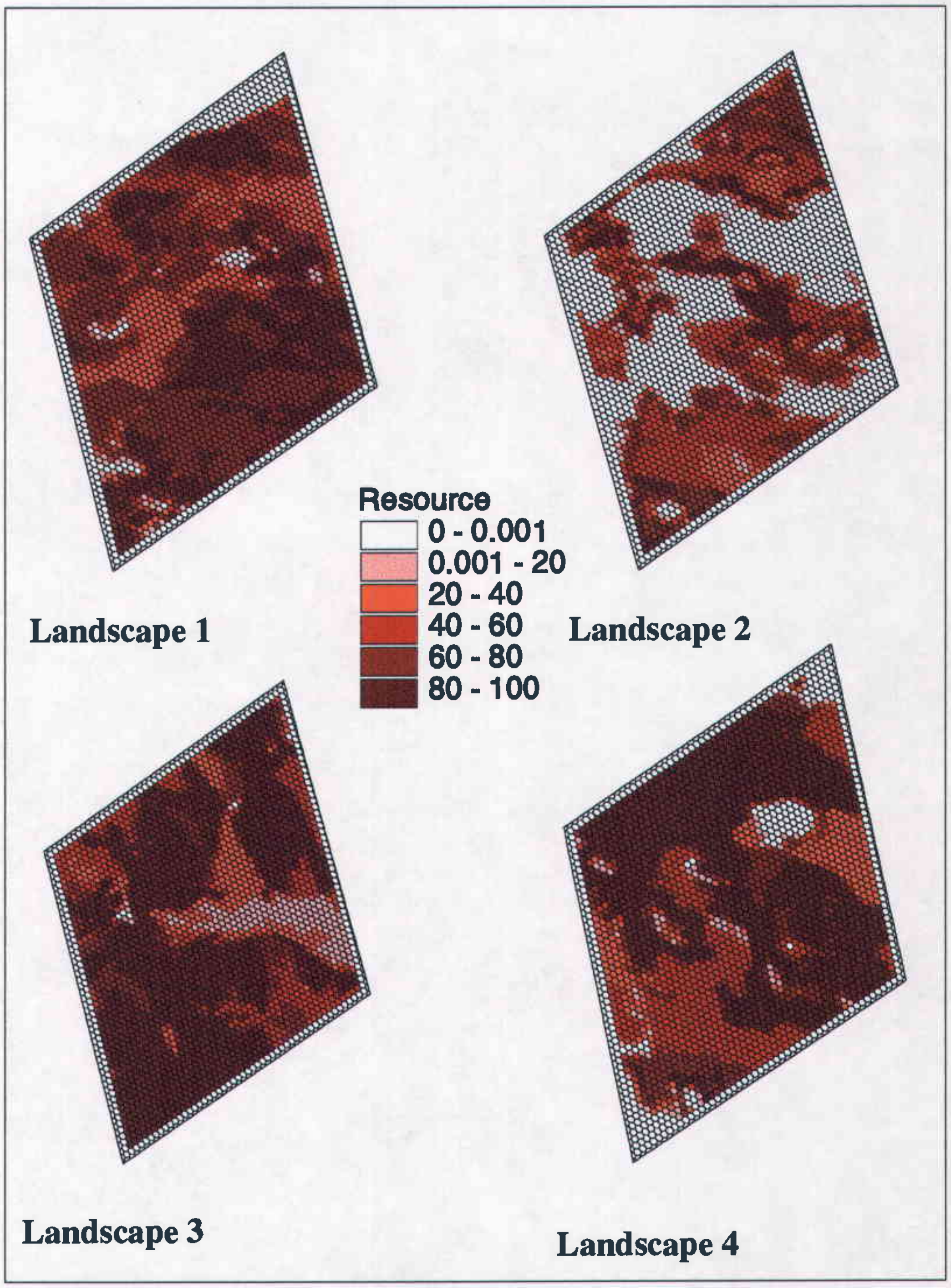


Figure 13. Hexagon map of resource in four sample landscapes (resolution 16)
The resource value of each hexagon is the average of resource values of all pixels belonging to it.The resource value of each pixel is calculated from Habitat Suitability Indices (HSI) of vegetation, soil, and elevation.
The higher the resource values, the more suitable the habitat for tortoises.

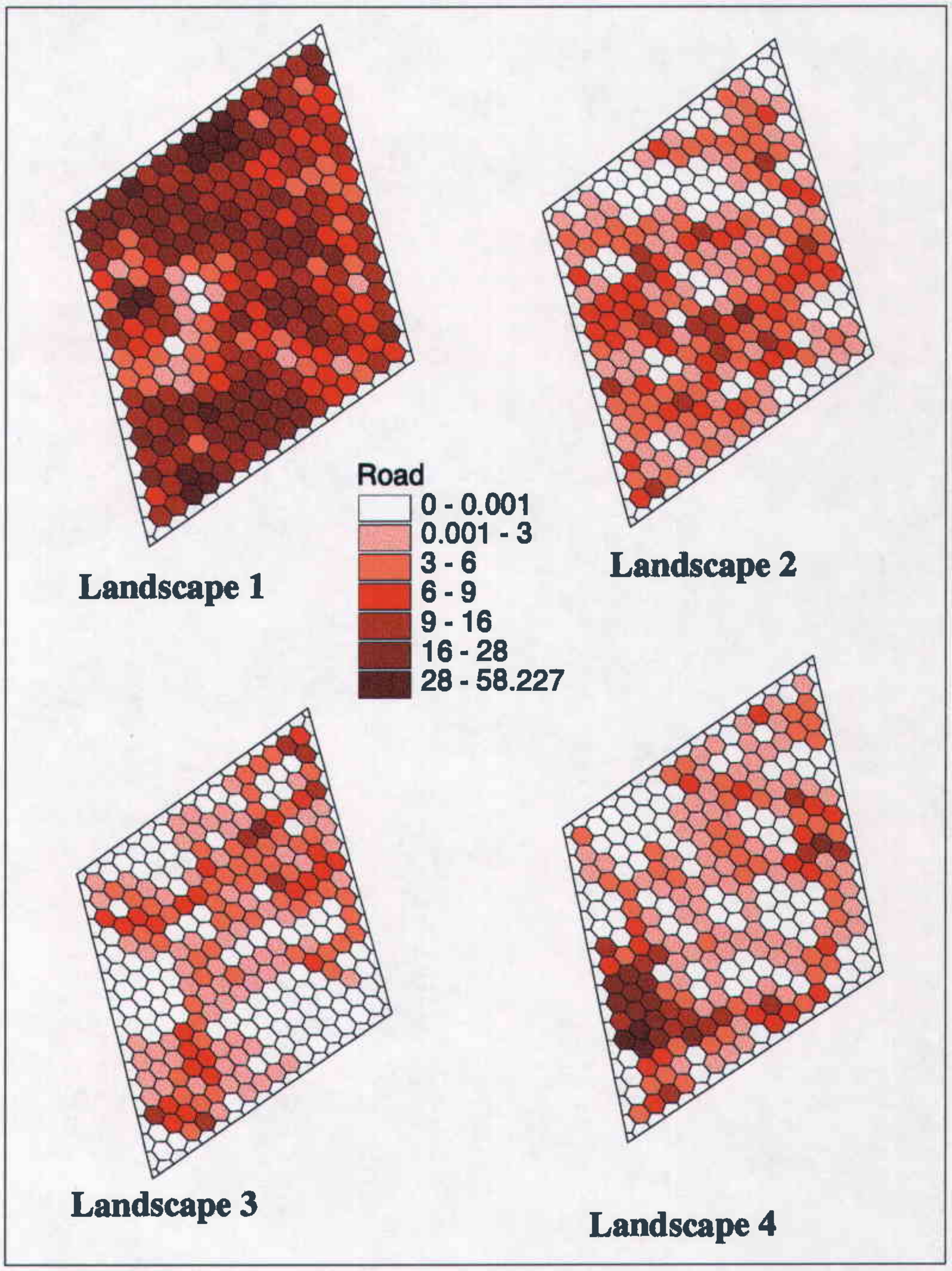


Figure 14. Hexagon map of roads in four sample landscapes (resolution 14)

Road value of each hexagon is the average of road values of all pixels belonging to it. Road value of each pixel is caculated from the weight of its road type, and rescaled to the range from 0 to 100. The higher the road values, the worse the habitat for tortoises.

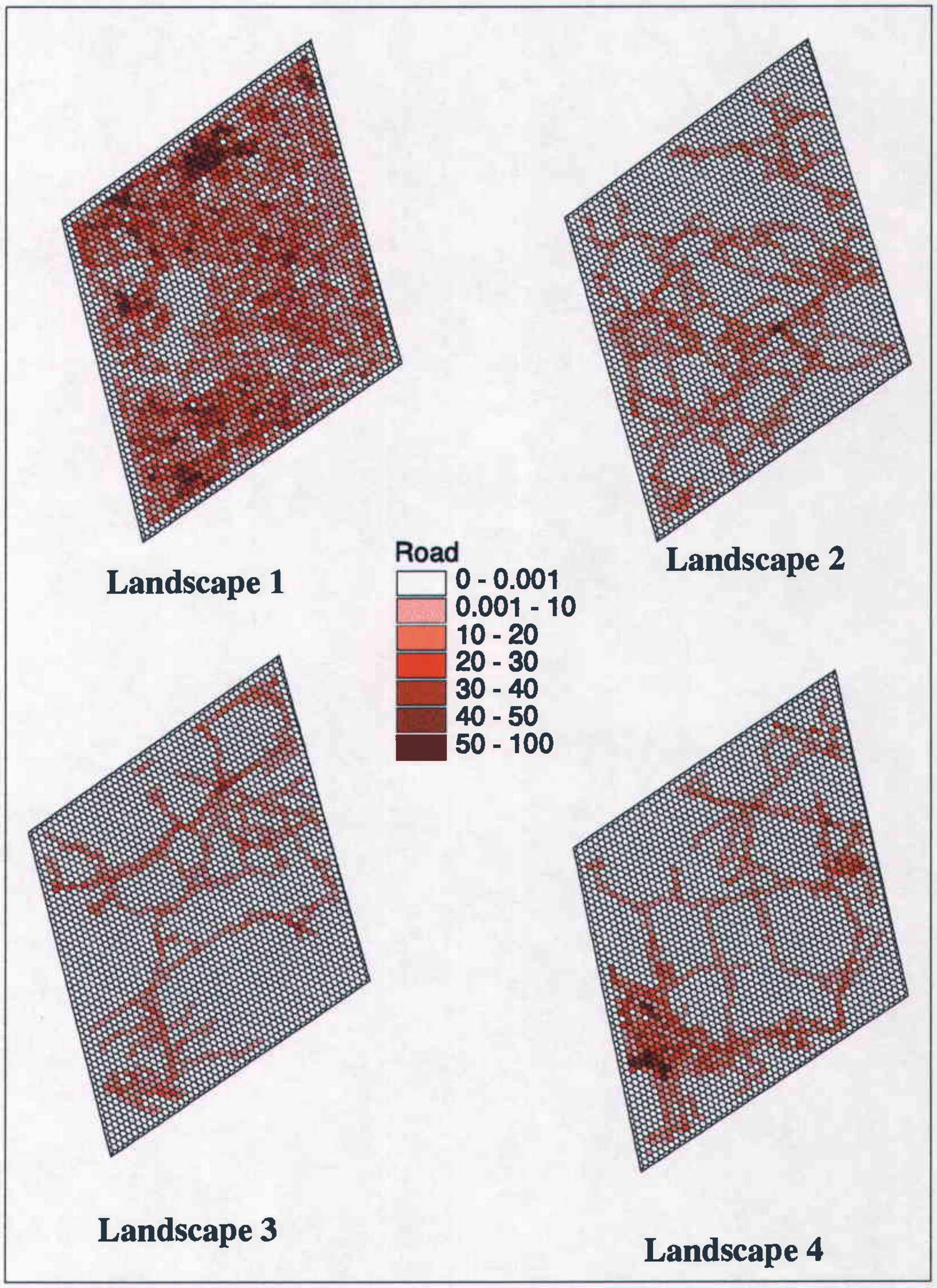


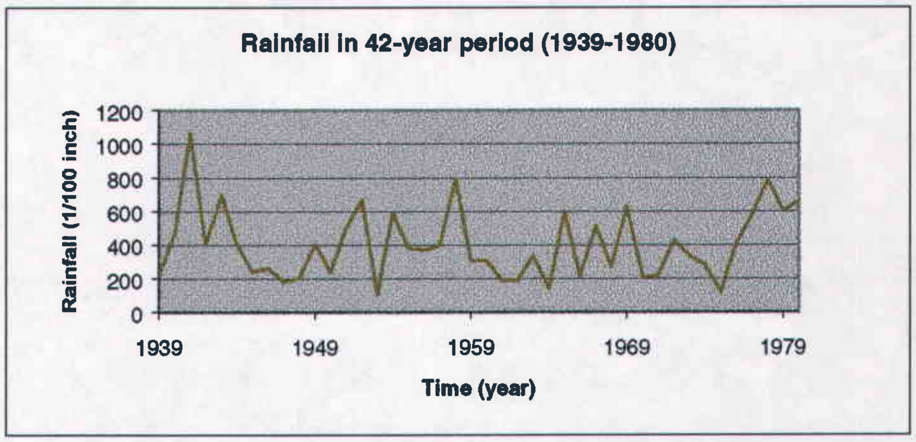
Figure 15. Hexagon map of roads in four sample landscapes (resolution 16)
Road value of each hexagon is the average of road values of all pixels belonging to it. Road value of each pixel is caculated from the weight of its road type, and rescaled to the range from 0 to 100. The higher the road values, the worse the habitat for tortoises.

Parameter combination : A1-A2-A3-A4-A5

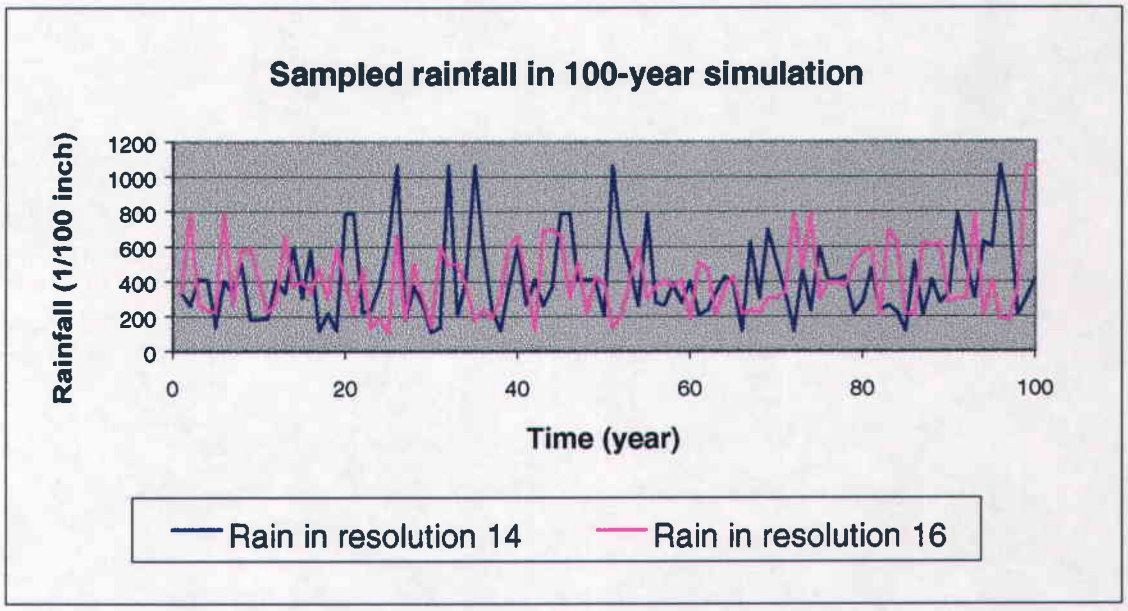
Parameter (name)	Possible values	Brief Description
A1 (rainPattern)	1	See Appendix B
	2	
A2 (mobilityPattern)	1	
	2	
A3 (movingEqua)	1	Constant a in Equa. (2) = 0.01
	2	Constant a in Equa. (2) = 0.03
	3	Constant a in Equa. (2) = 0.06
	4	Constant a in Equa. (2) = 0.10
A4 (RepSurEquaRoad)	0	Constant a in Equa. (5) = 0
	1	Constant a in Equa. (5) = 0.01
	2	Constant a in Equa. (5) = 0.02
	3	Constant a in Equa. (5) = 0.03
	4	Constant a in Equa. (5) = 0.04
A5 (RepSurEquaRain)	0	Constant b in Equa. (6) = 0
	1	Constant b in Equa. (6) = 0.01
	2	Constant b in Equa. (6) = 0.02
	3	Constant b in Equa. (6) = 0.03
	4	Constant b in Equa. (6) = 0.04

Table 3. Description for parameter combination used in the model's simulations

different landscapes of different characteristics, and to test the sensitivity of model parameters. Since the purpose of the model is not to predict the tortoise population in reality, all the simulations are run with the same random seed, which means all simulations with the same grid resolution have the same temporal pattern of rainfall (Figure 16). All parameter combinations are run for both grid resolution 14 and 16 for each landscape. With the temporal pattern of rainfall held constant, all other comparisons represent the difference in model parameters, and the results truly reflect the change of model parameters or the difference of habitat patterns. The total population and population growth rate lambda in four sample landscapes are recorded for every time step, and results from different parameter combinations are drawn from Figures 17 to 25 (see Table 3 for description about parameter combination in detail). First (Figures 17 - 20), two parameter combinations, 1-1-3-3-3 and 2-2-2-2-2, are used to run simulations. The five parameters represent from left to right spatial rain pattern, the mobility pattern of the eight size classes, moving equation, equation of road effect on survival and reproduction, and equation of rain effect on survival and reproduction (see Model Structure and Table 3). The results from these two parameter combinations in both grid resolutions show the same trend of population dynamics in the four landscapes. Population declines very fast in landscape 1 with high road value, and population growth rate lambda is smallest in this landscape. Lambda values in three other landscapes are very close to one another. Population in landscape 2 is always lower, since this landscape has limited resources. Lambda values for these three landscapes are lower with combination 2-2-2-2-2 than with combination 1-1-3-3-3, one shows a decline in population, and the other shows a stable population.

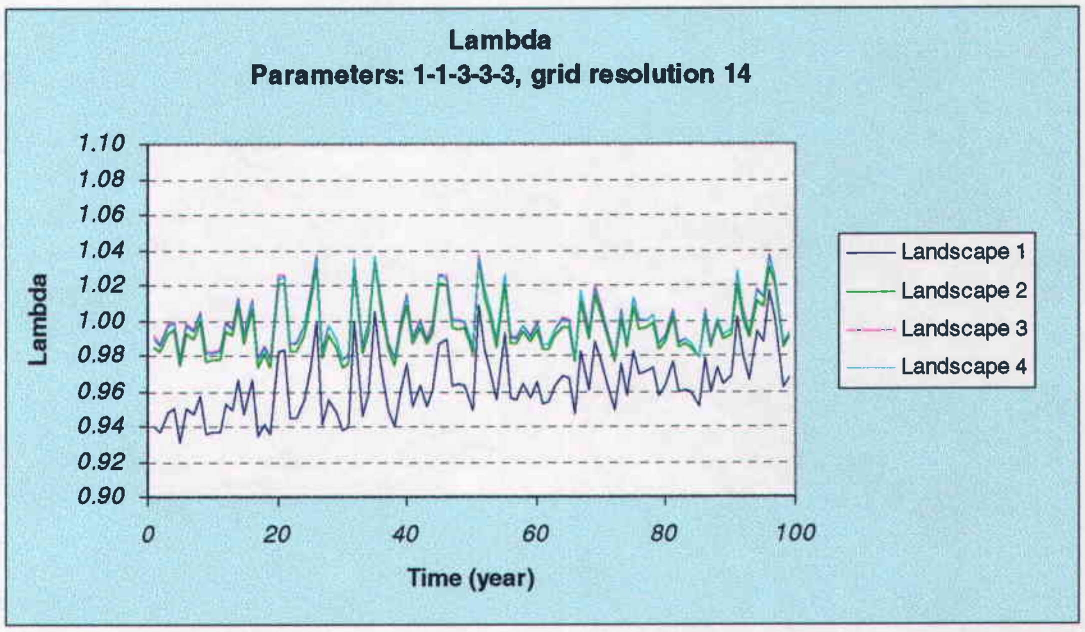


(a)

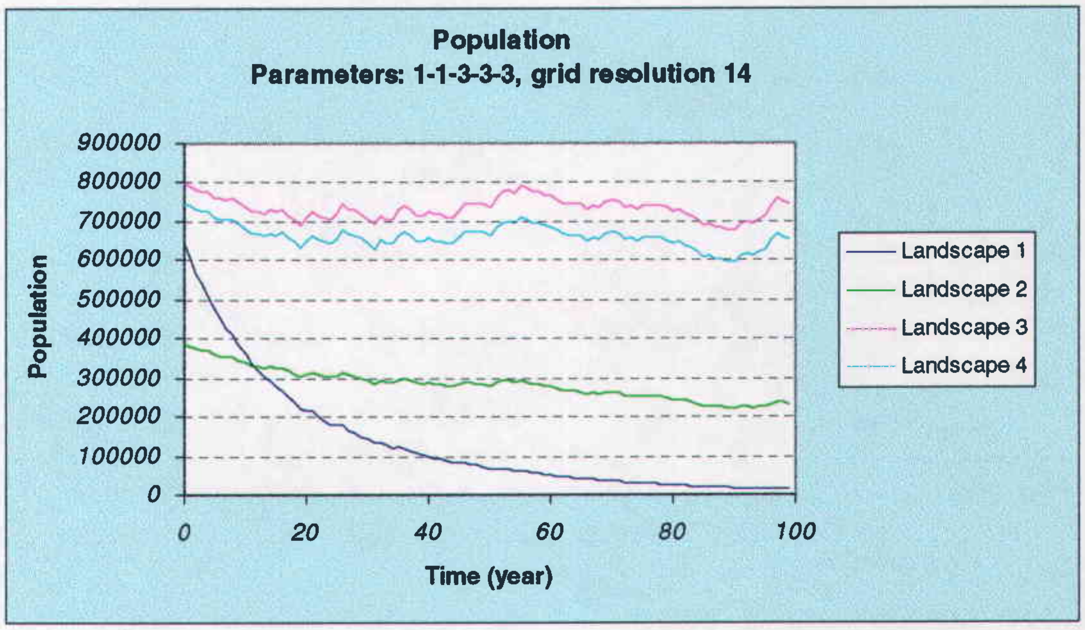


(b)

Figure 16. (a) Annual rainfall in 42 year period (1939-1980) and (b) rainfall sampled in simulations with grid resolutions 14 and 16. Simulations used the same random seed.

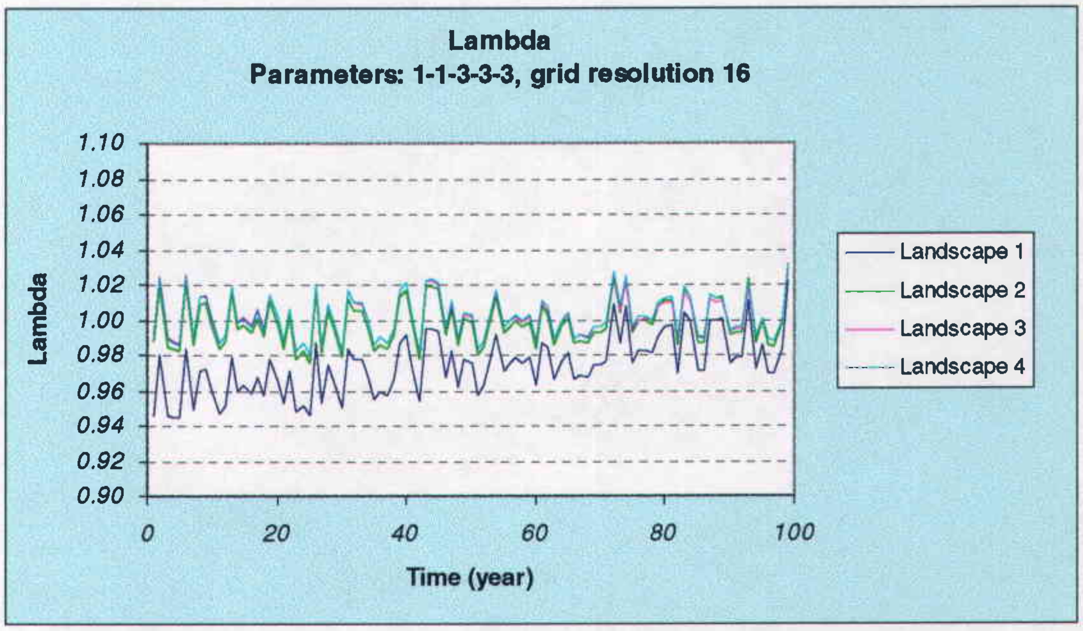


(a)

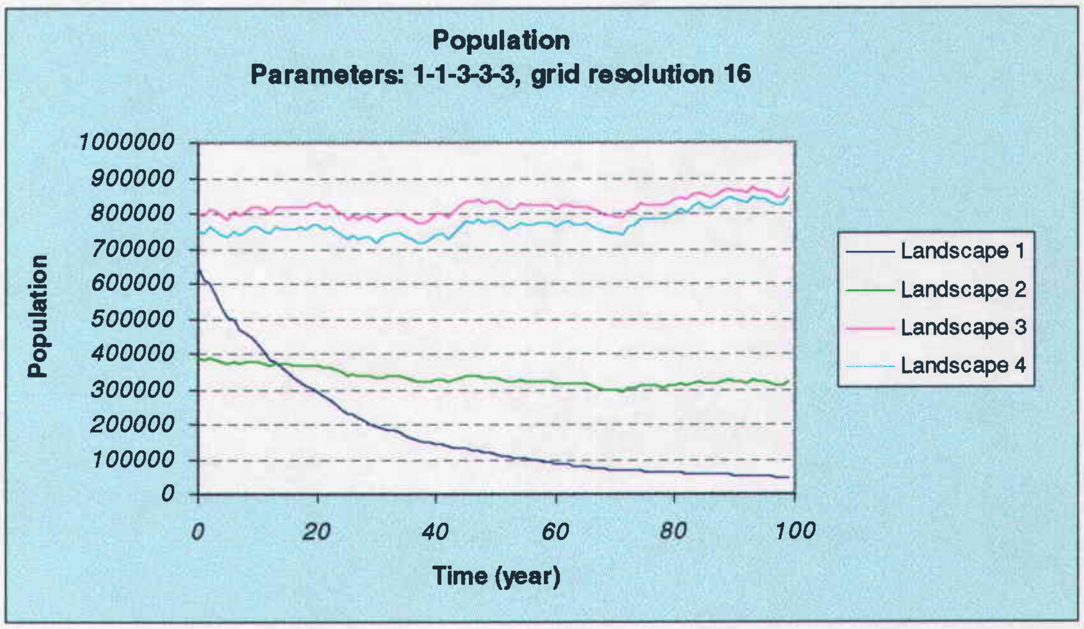


(b)

Figure 17. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 1-1-3-3-3, grid resolution 14

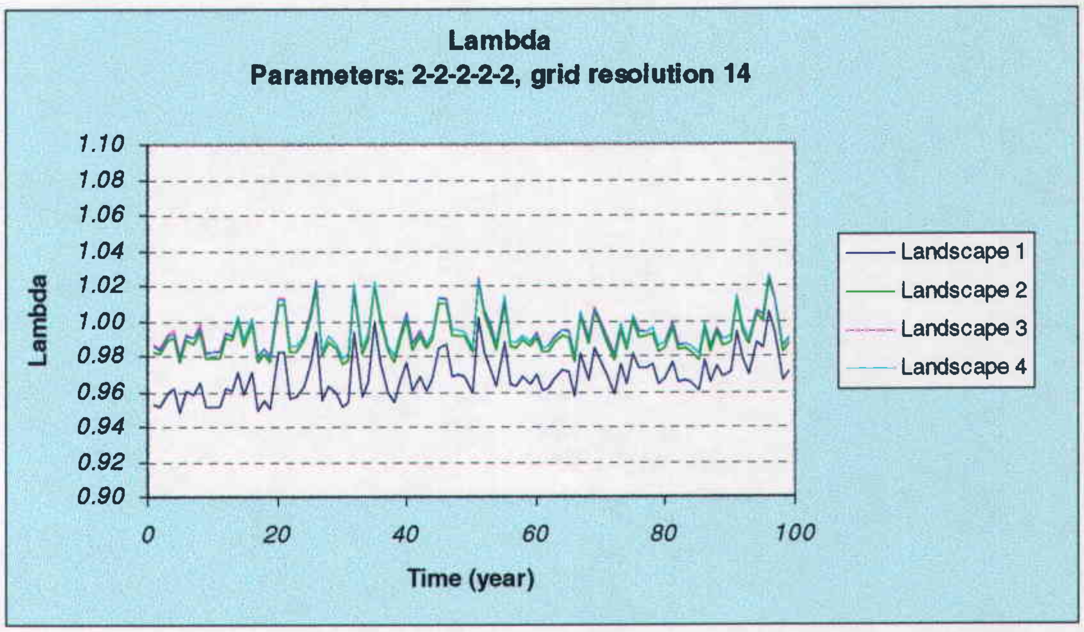


(a)

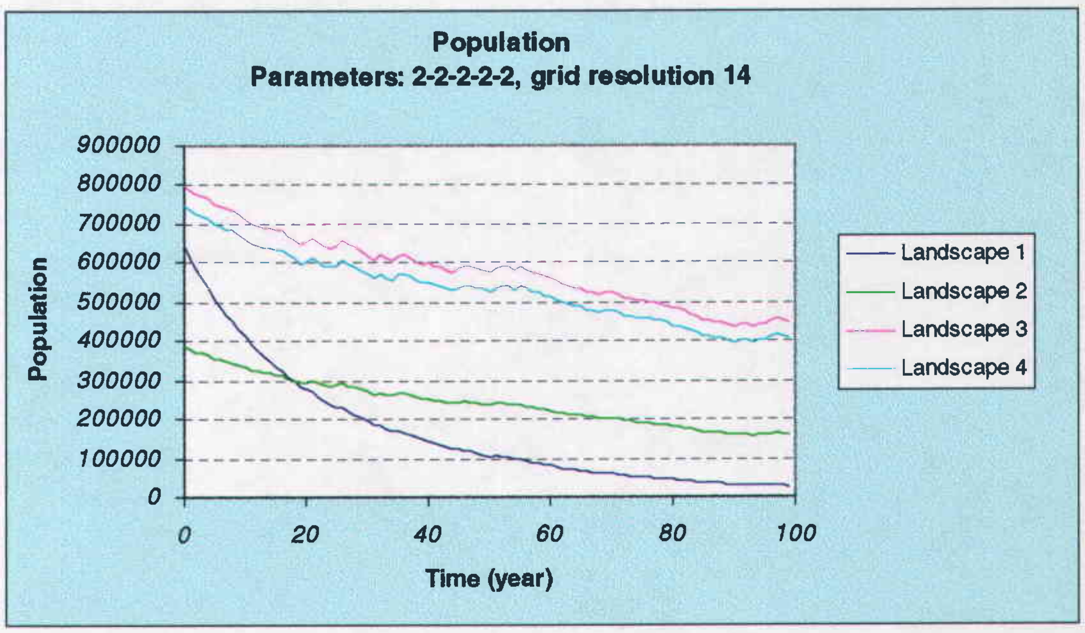


(b)

Figure 18. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 1-1-3-3-3, grid resolution 16

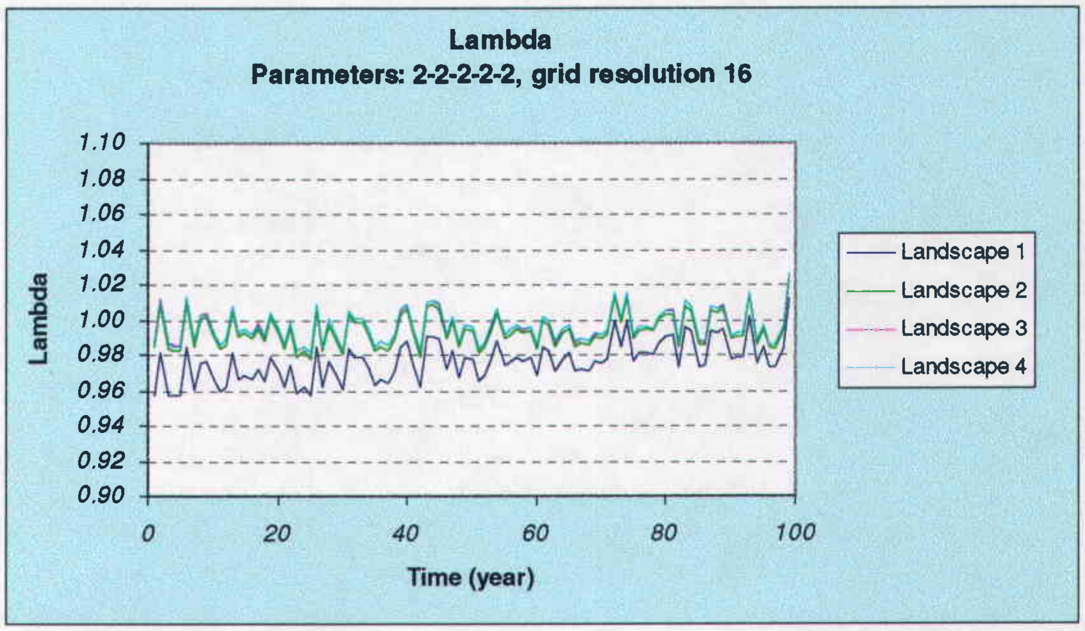


(a)

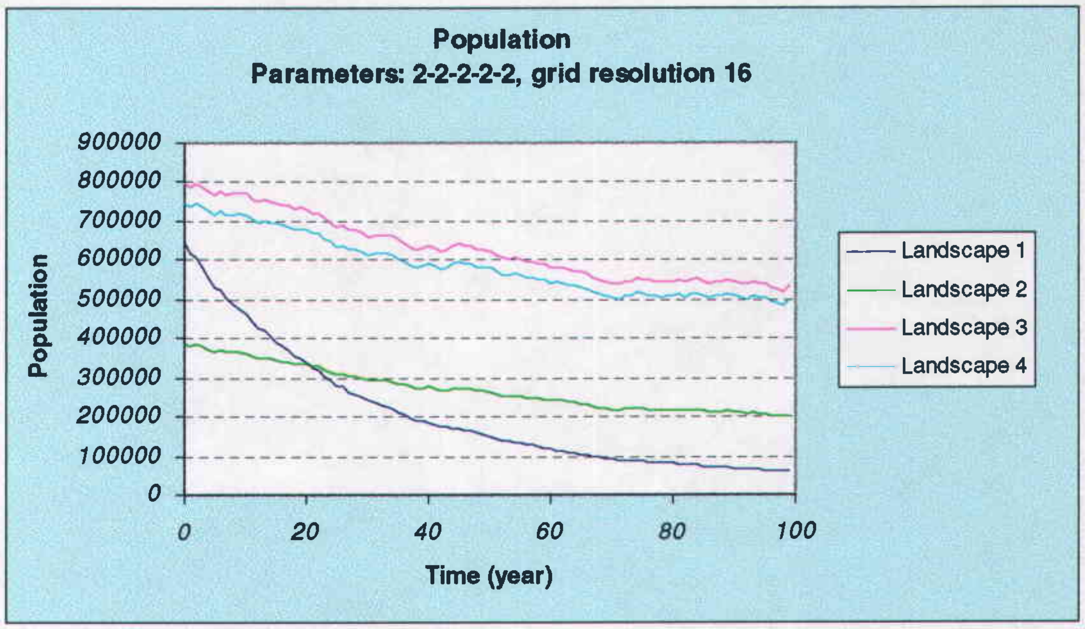


(b)

Figure 19. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 2-2-2-2-2, grid resolution 14



(a)

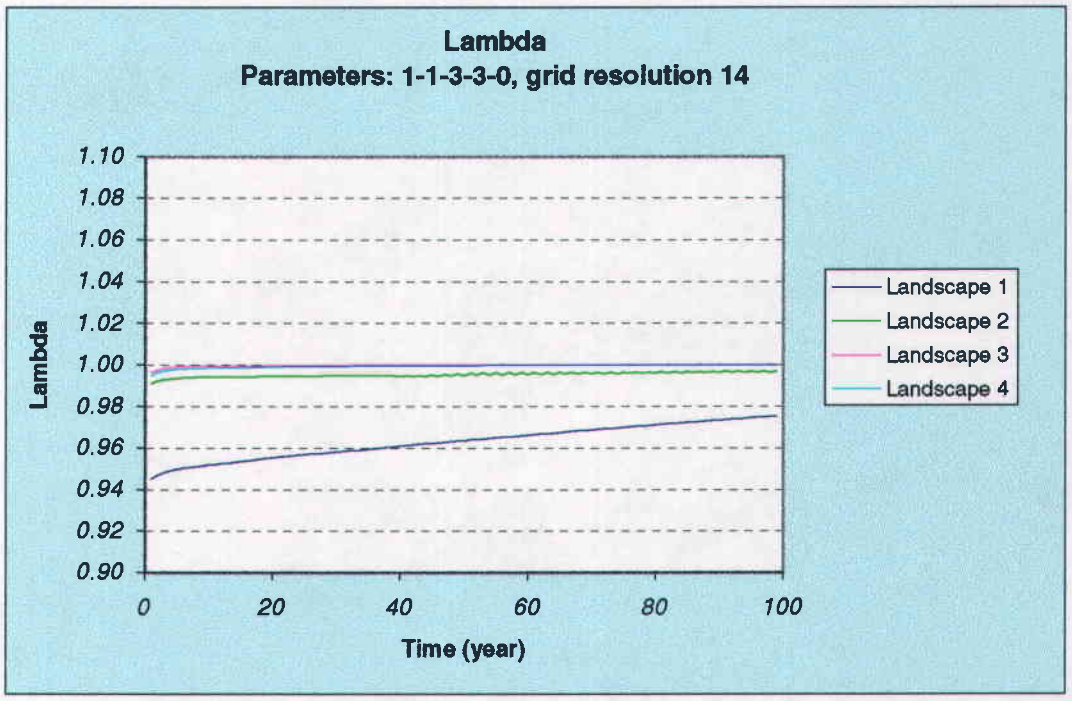


(b)

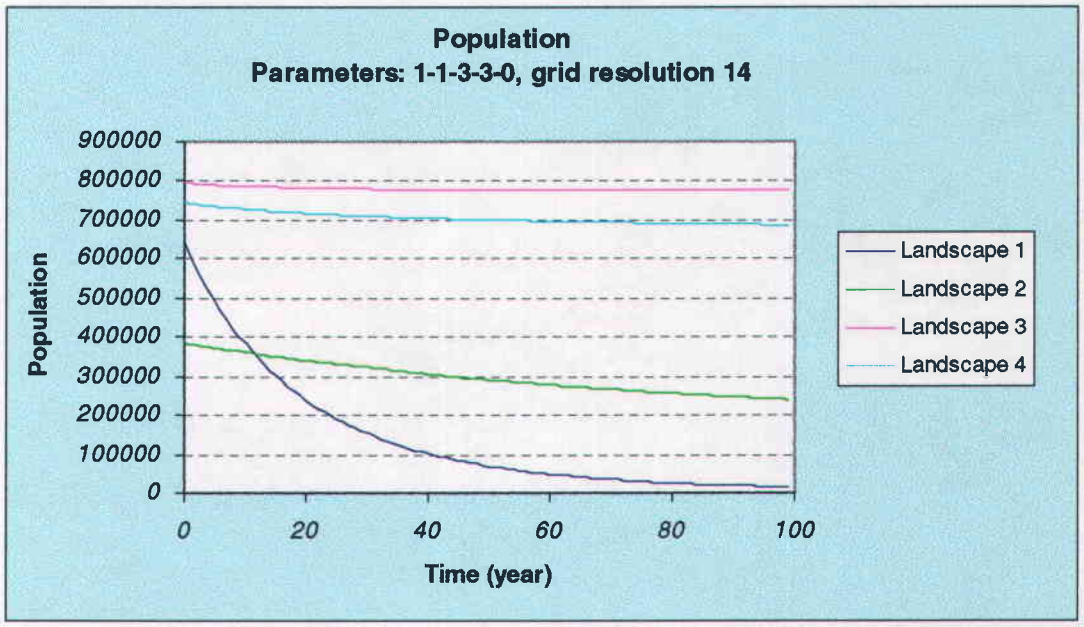
Figure 20. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 2-2-2-2-2, grid resolution 16

Next (Figures 21-23), I studied the effect of roads alone by setting the rain effect to zero (combinations 1-1-3-3-0, and 2-2-2-2-0). The results show the high correlation between road value and population trends. Populations in landscape 1 with a high road value decrease over time. Three other landscapes have the same pattern of population dynamics with a relatively stable population. The results in two grid resolutions for combination 2-2-2-2-0 do not show any obvious difference, so that for this combination only results for resolution 14 are presented. For combination 1-1-3-3-0, lambda values of landscape 3 and 4 are close to one at resolution 14, but slightly higher than one at resolution 16, so that one shows a stable population and the other shows an increase in population.

Figures 24 - 25 are the results from simulations with the rain effect alone (combinations 1-1-3-0-3 and 2-2-2-0-2). Since there is no obvious difference between the two grid resolutions, only results at resolution 14 are presented. Since the variation of lambda in all four landscapes is the response to variation of rainfall alone and all the runs have the same random seed, lambda values in all four landscapes are the same every year. Lambda values vary around lambda of the static projection matrix (0.982), and all four landscapes show a decline in population.

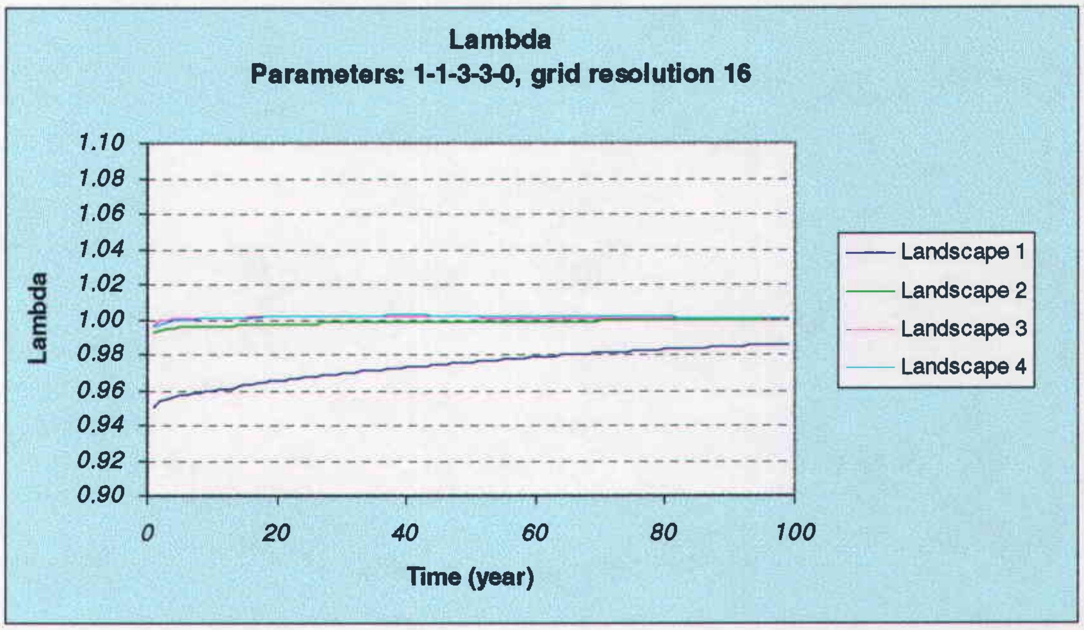


(a)

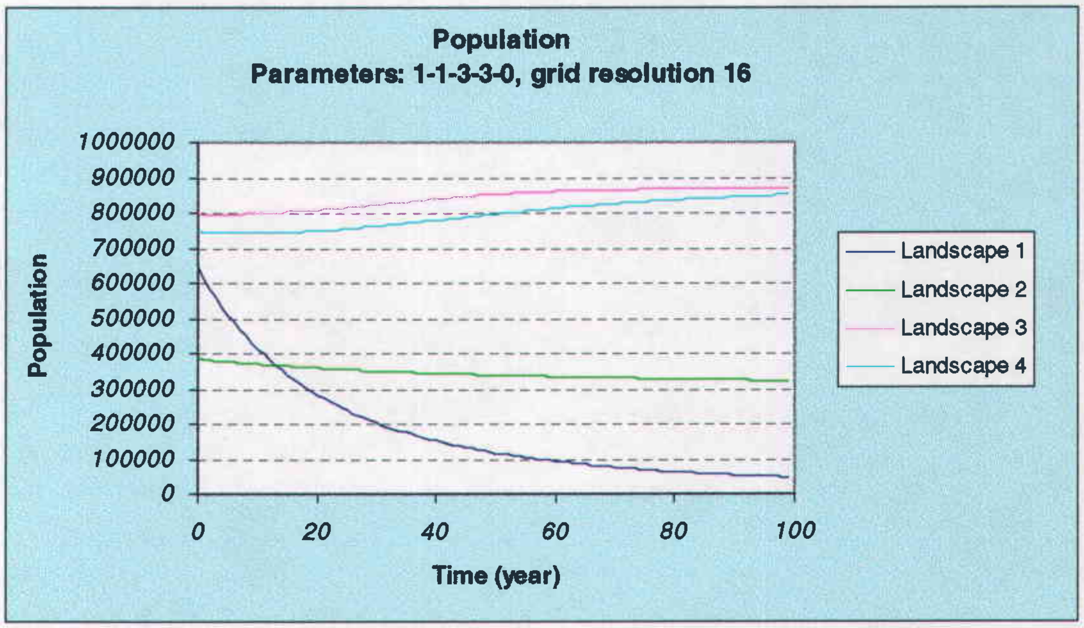


(b)

Figure 21. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 1-1-3-3-0, grid resolution 14

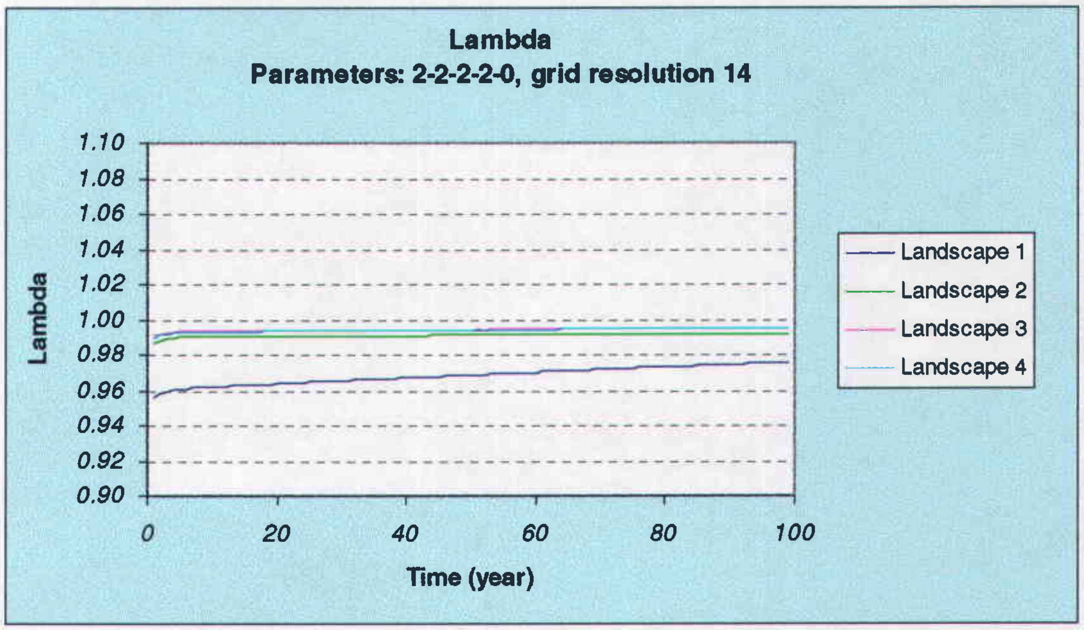


(a)

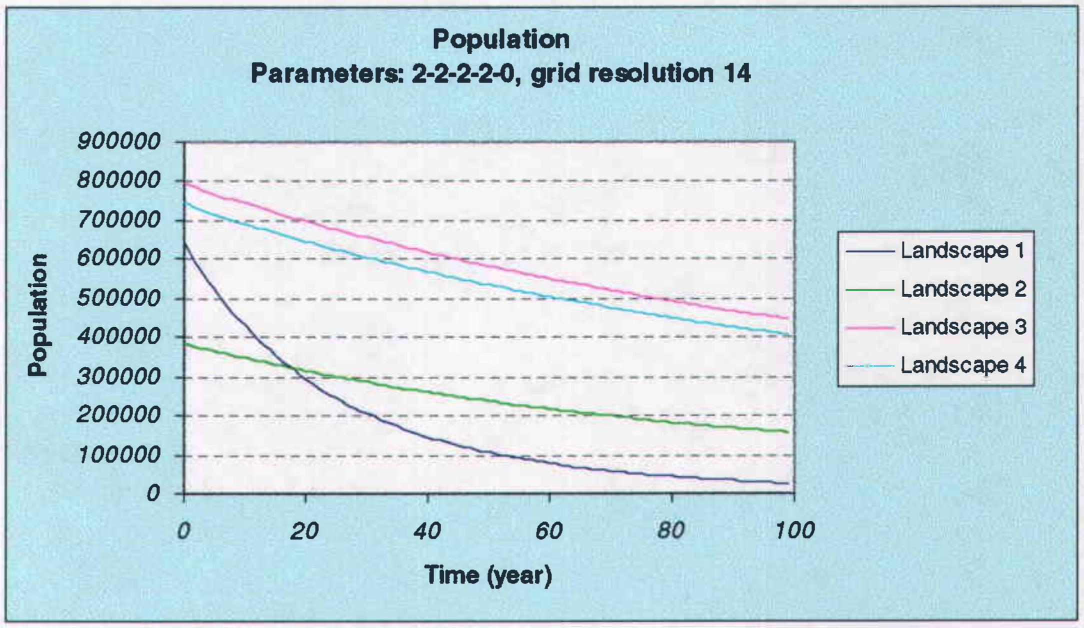


(b)

Figure 22. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 1-1-3-3-0, grid resolution 16

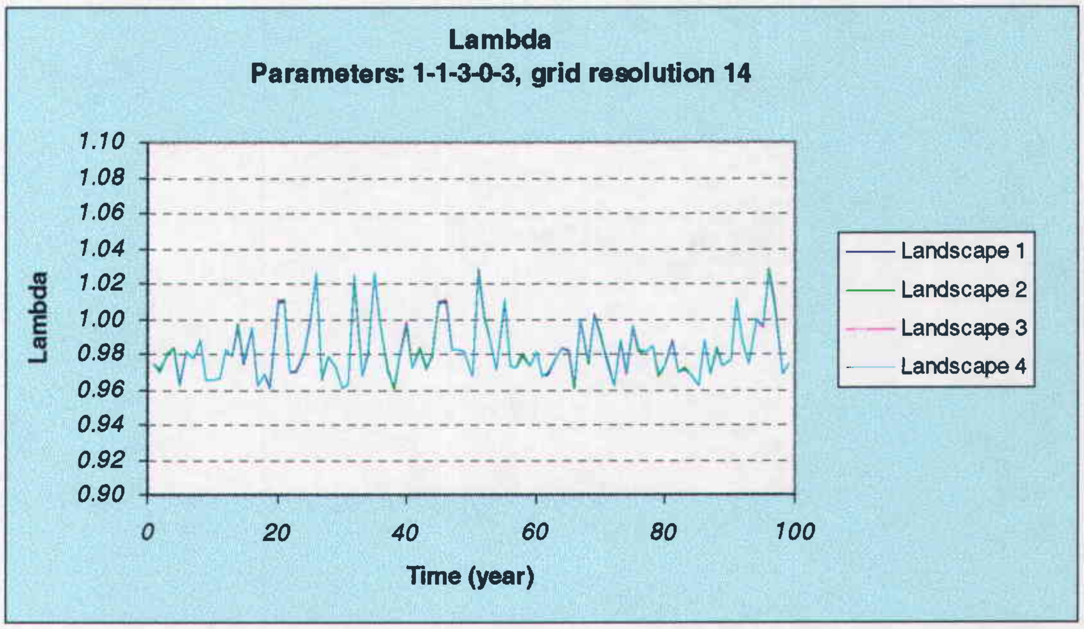


(a)

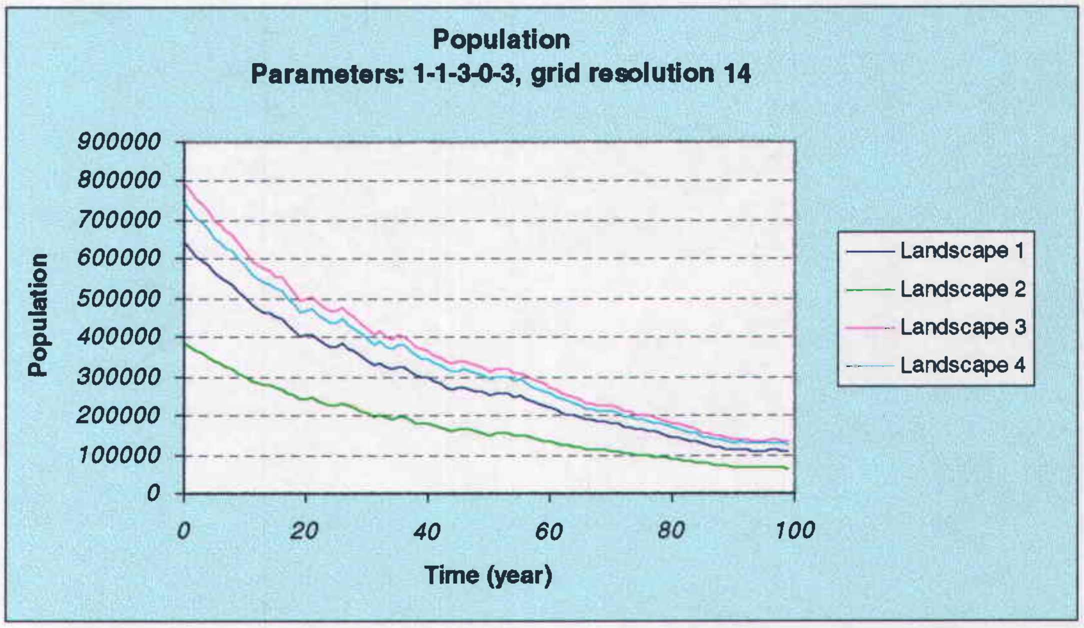


(b)

Figure 23. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 2-2-2-2-0, grid resolution 14

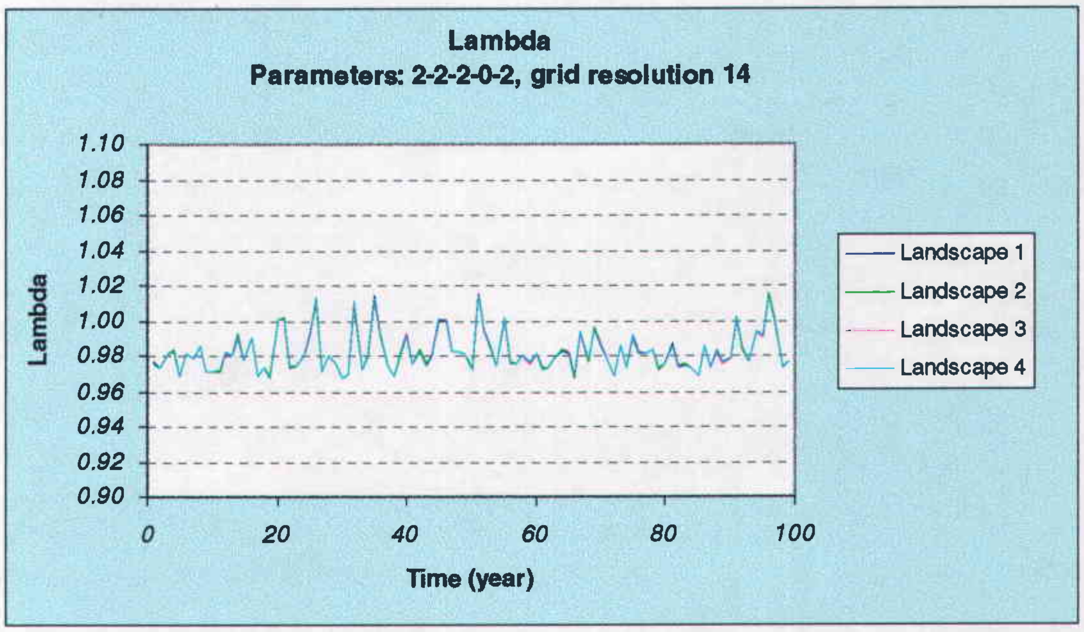


(a)

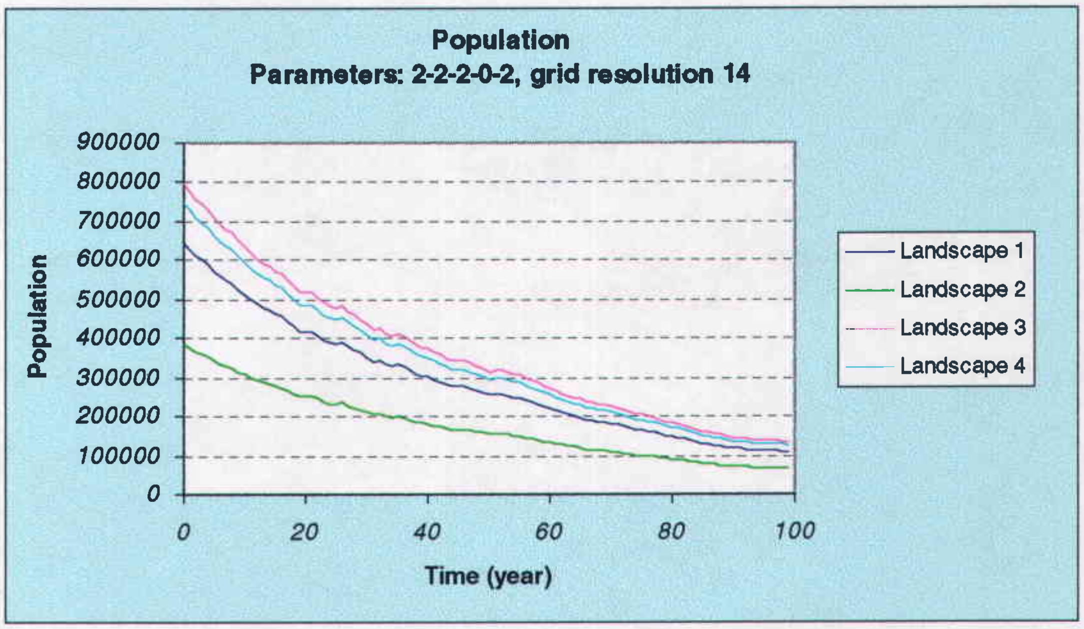


(b)

Figure 24. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 1-1-3-0-3, grid resolution 14



(a)



(b)

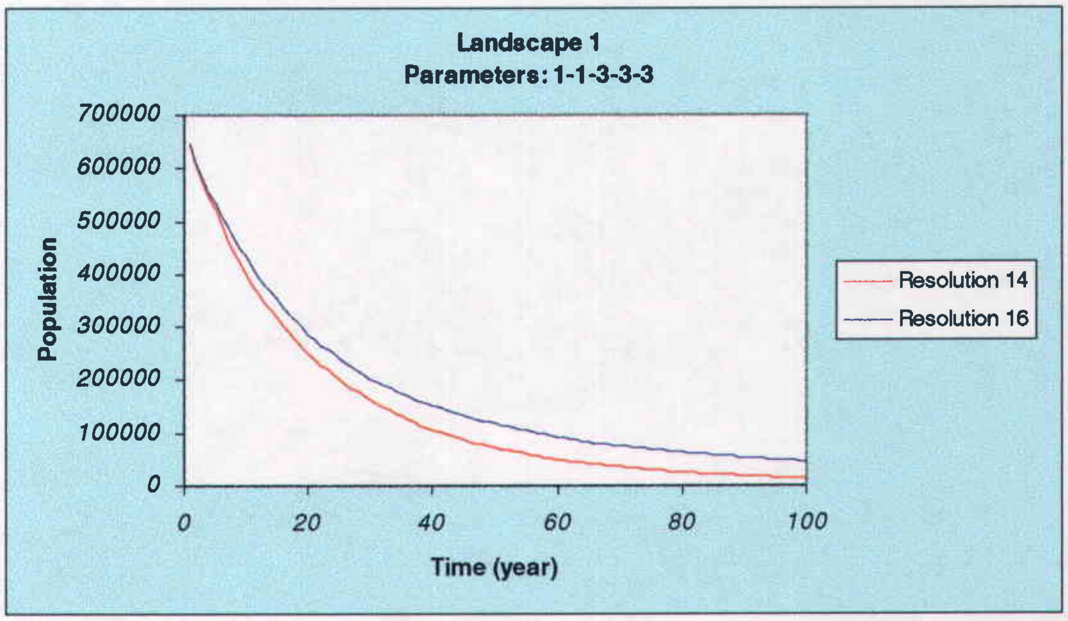
Figure 25. Population growth rate lambda (a) and total population (b) in four sample landscapes – parameters: 2-2-2-0-2, grid resolution 14

3.3. Results from Sensitivity Analysis

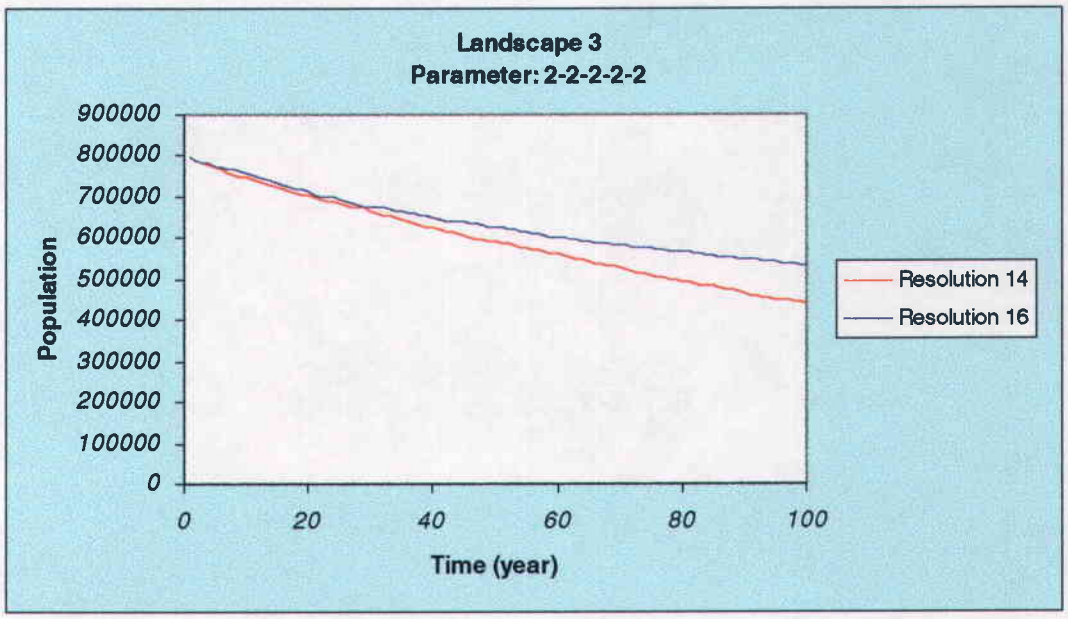
In this section, the results from sensitivity testing of model parameters are discussed. In the model, the parameters controlling model results are grid resolution, spatial distribution of rain every year, mobility pattern of individuals in different size classes, mobility of species (moving equation), and effect of road on survival and reproduction (survival and reproduction equation for road effect), effect of temporal variation of rain on survival and reproduction (survival and reproduction equation for rain effect). The sensitivity of each parameter on model results is tested by varying it within the proper range or setting it to possible values while holding all other parameters constant.

Results from the sensitivity of grid resolutions 14 and 16 are shown in Figure 26. Since the same series of temporal variation in rainfall at two grid resolutions cannot be generated by running simulations with one random seed, the lines are the average from twenty replicate runs with twenty different random seeds for both resolutions. Simulations are run with parameter combination 1-1-3-3-3 for landscape 1, and 2-2-2-2-2 for landscape 3. The results show a higher population at resolution 16 than at resolution 14 in both cases.

I used two control parameter combinations 1-1-3-3-3 and 2-2-2-2-2 to test the sensitivity of all other parameters, which means that those numbers are held constant while varying each parameter separately. Results are presented in Figure 27-34. The population with the control parameter combination has the red color. The variations by different road effects, by different rain effects, and by different characteristics of species mobility (moving equation) are drawn in a blue, magenta, and teal (aquamarine) color, respectively. Effect of size-specific mobility is the yellow line, and the effect of

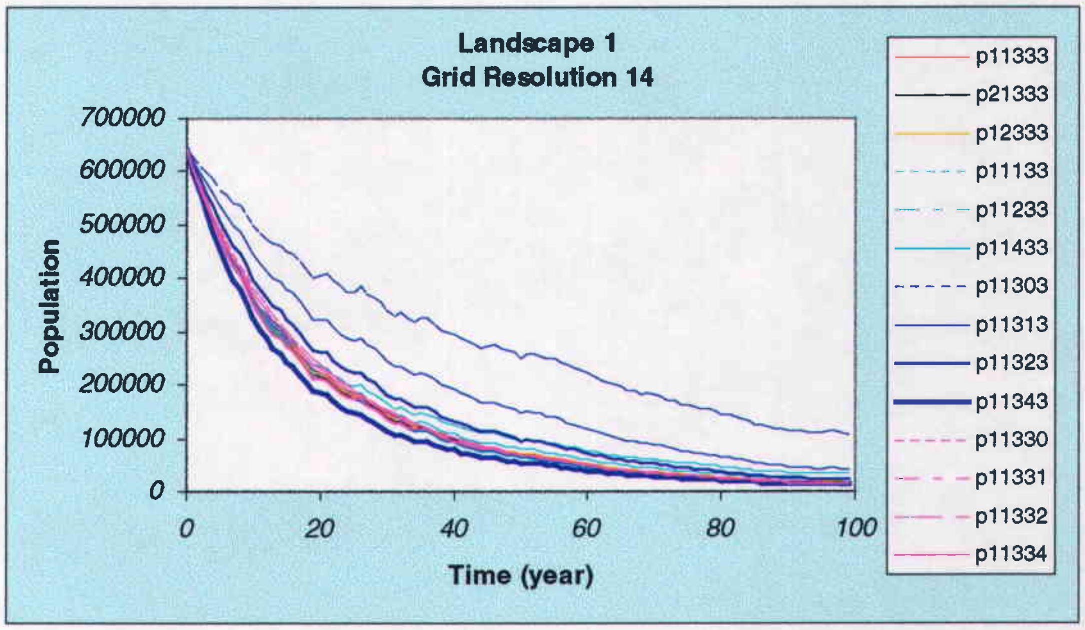


(a)

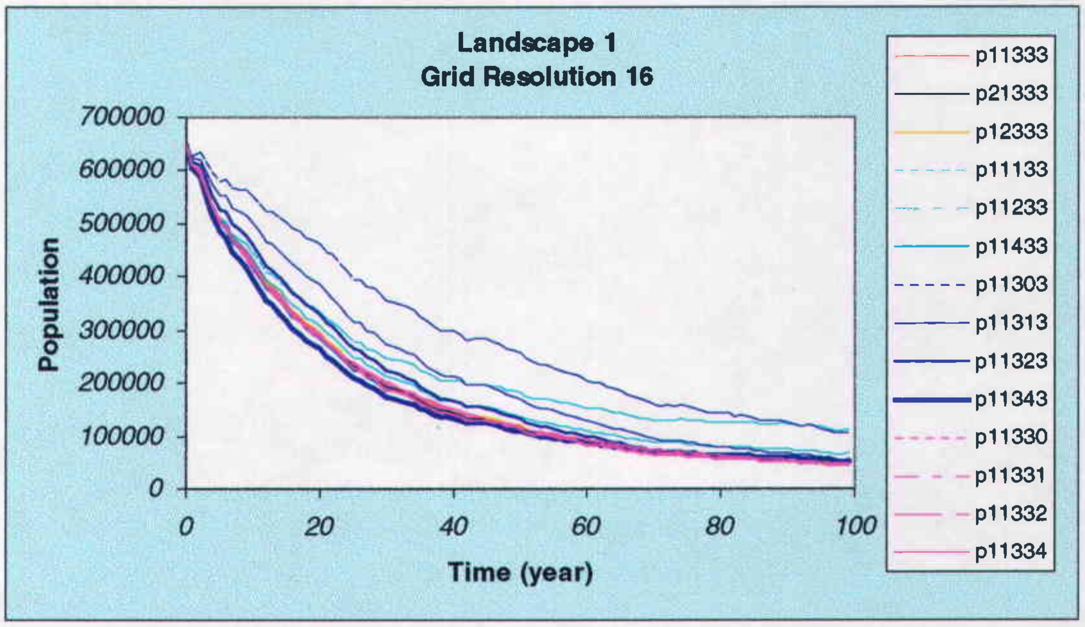


(b)

Figure 26. Projected population for (a) landscape 1, parameters: 1-1-3-3-3 and (b) landscape 3 with parameters: 2-2-2-2-2. Results are from twenty replicate runs.

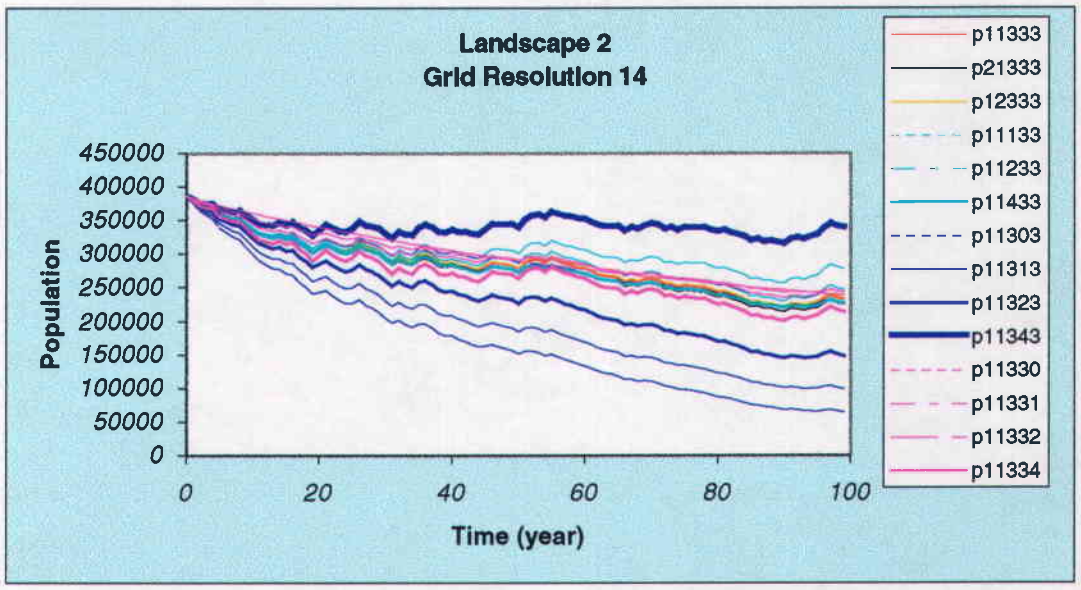


(a)

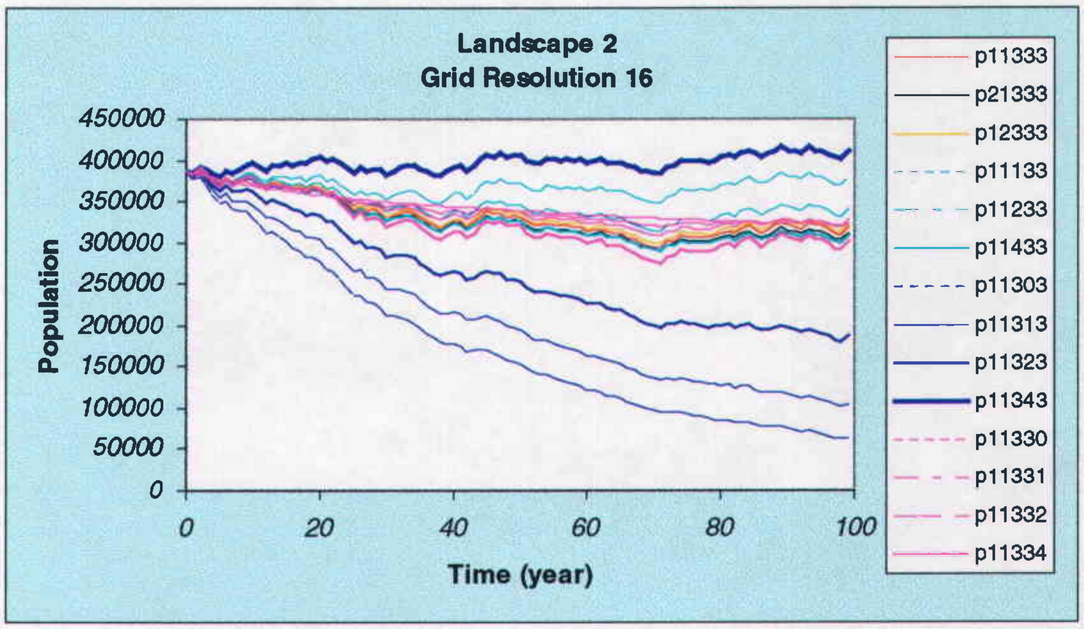


(b)

Figure 27. Projected population for landscape 1 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 1-1-3-3-3.

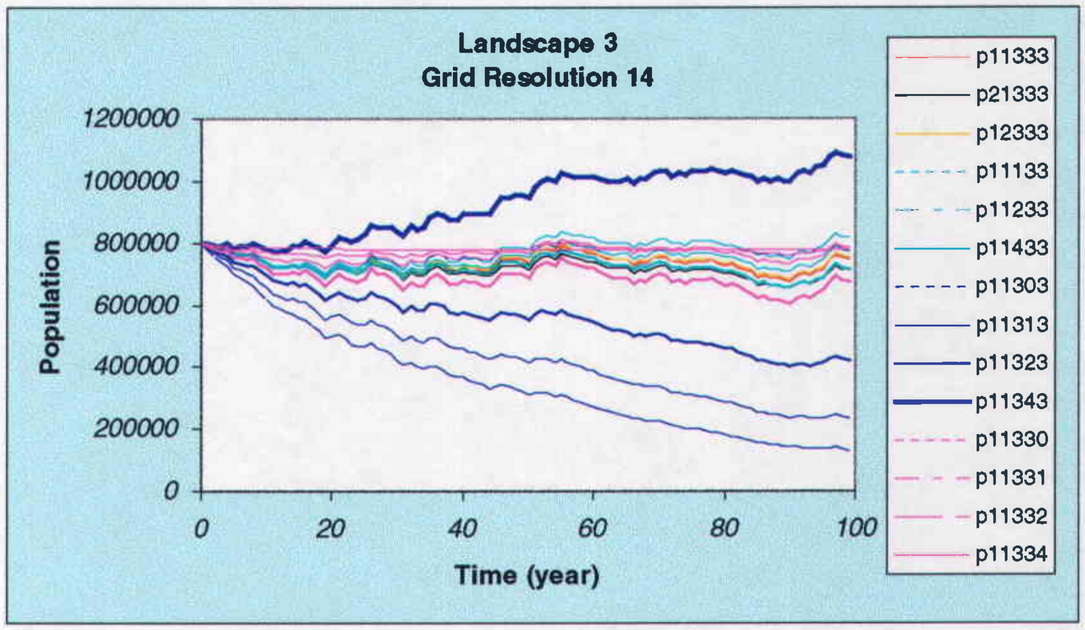


(a)

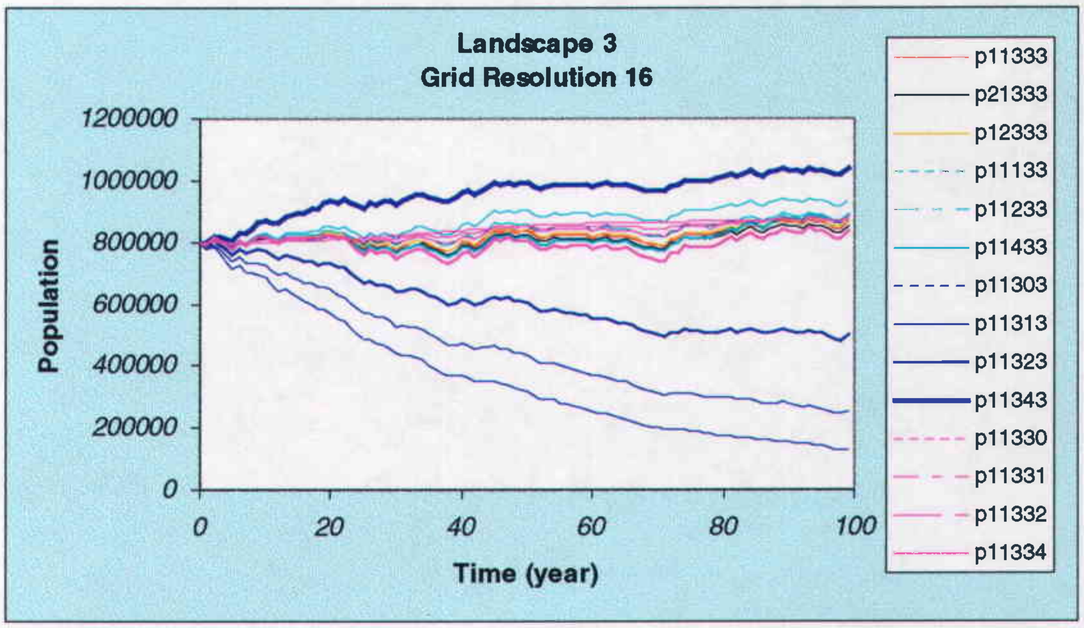


(b)

Figure 28. Projected population for landscape 2 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 1-1-3-3-3.

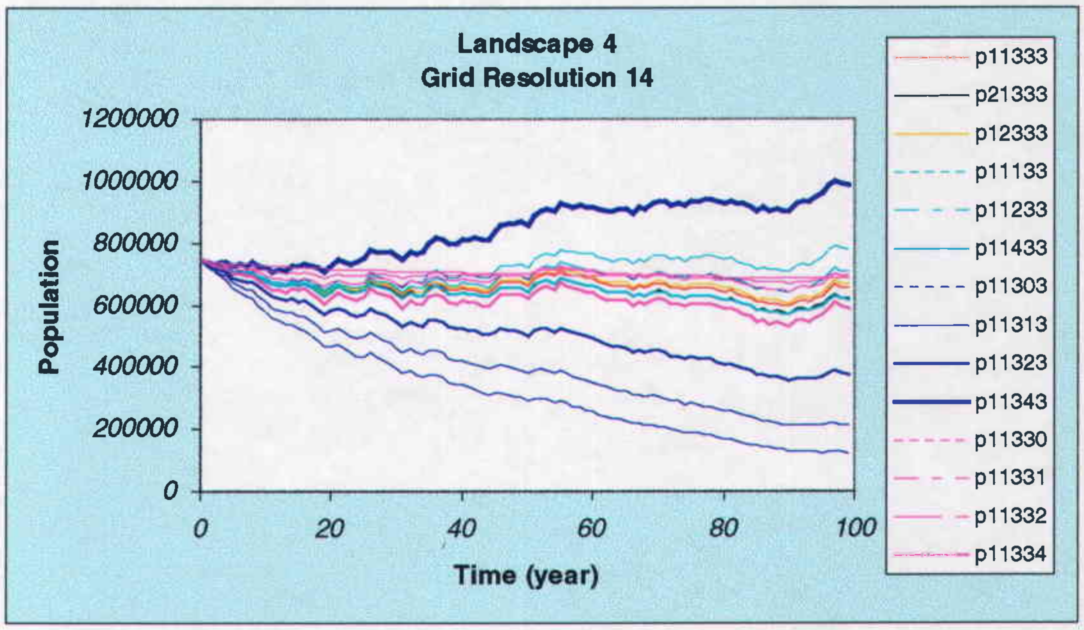


(a)

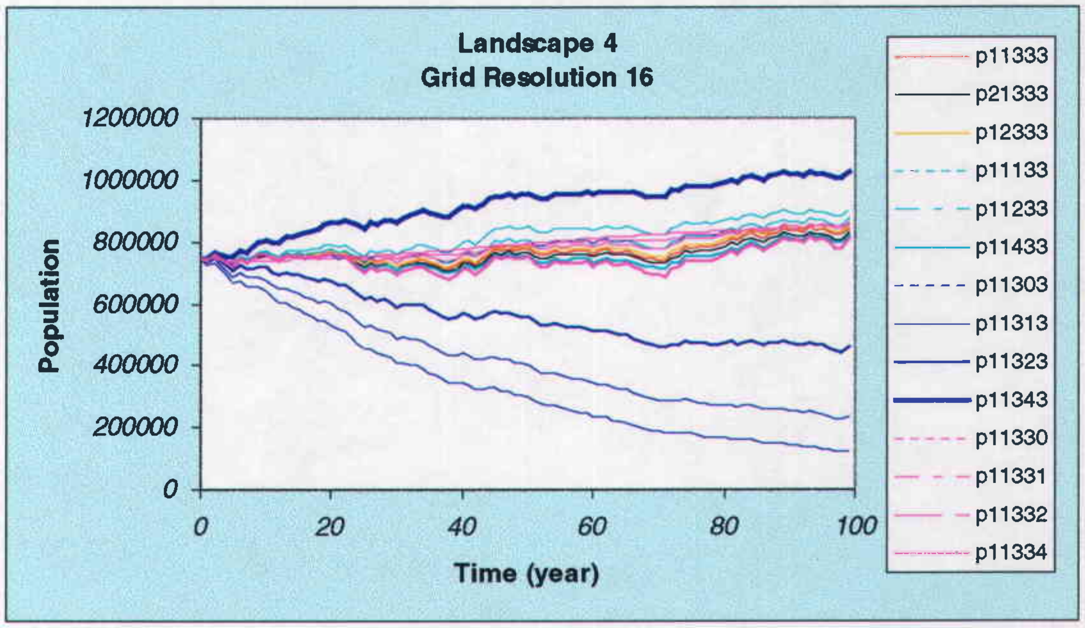


(b)

Figure 29. Projected population for landscape 3 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 1-1-3-3-3.

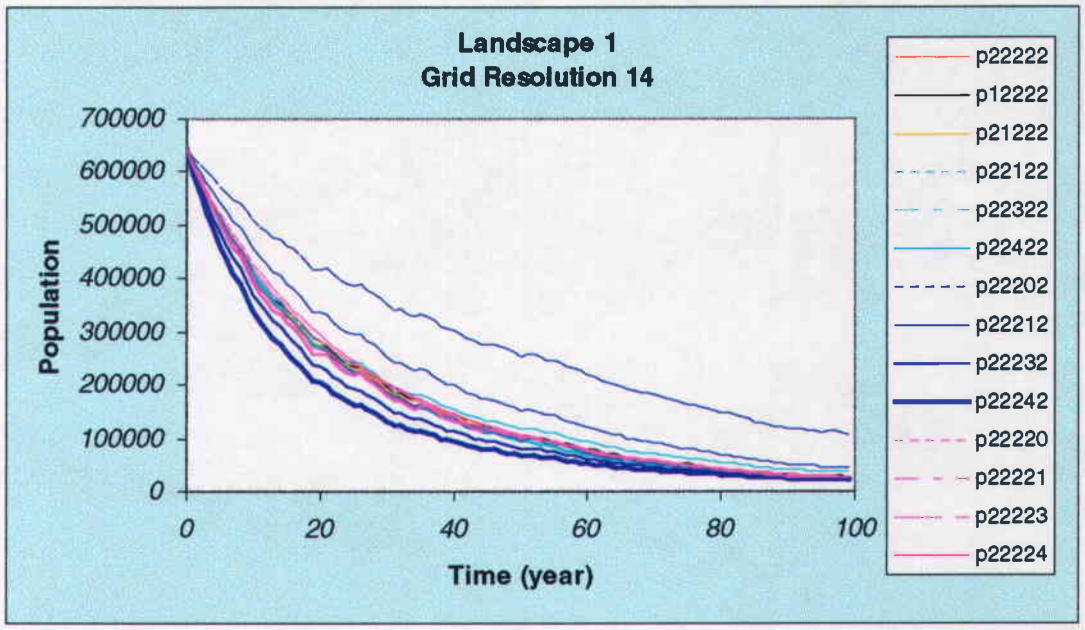


(a)

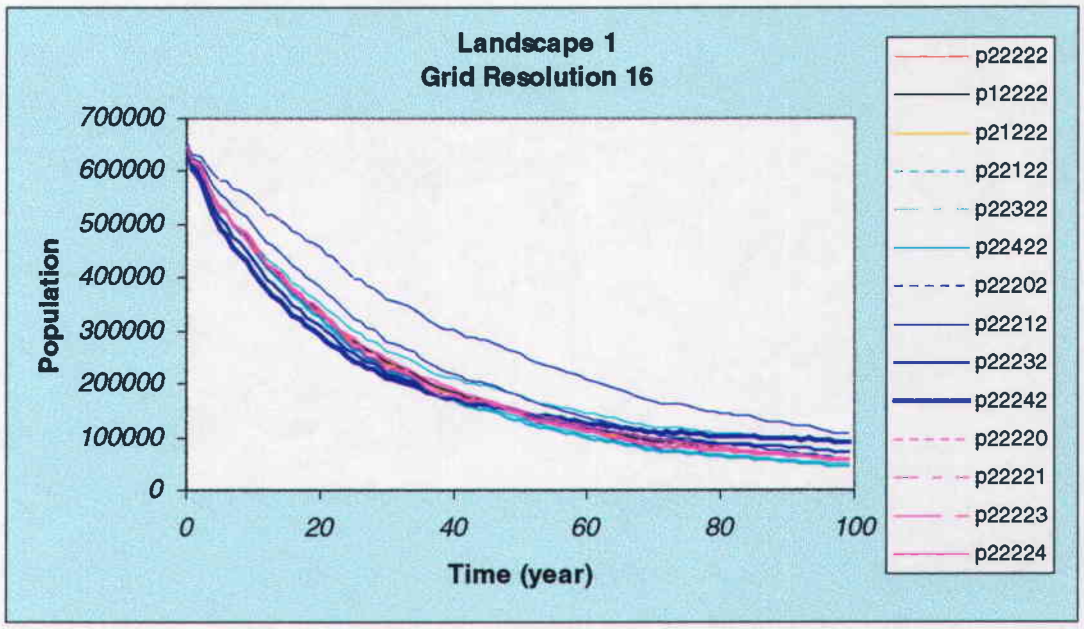


(b)

Figure 30. Projected population for landscape 4 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 1-1-3-3-3.

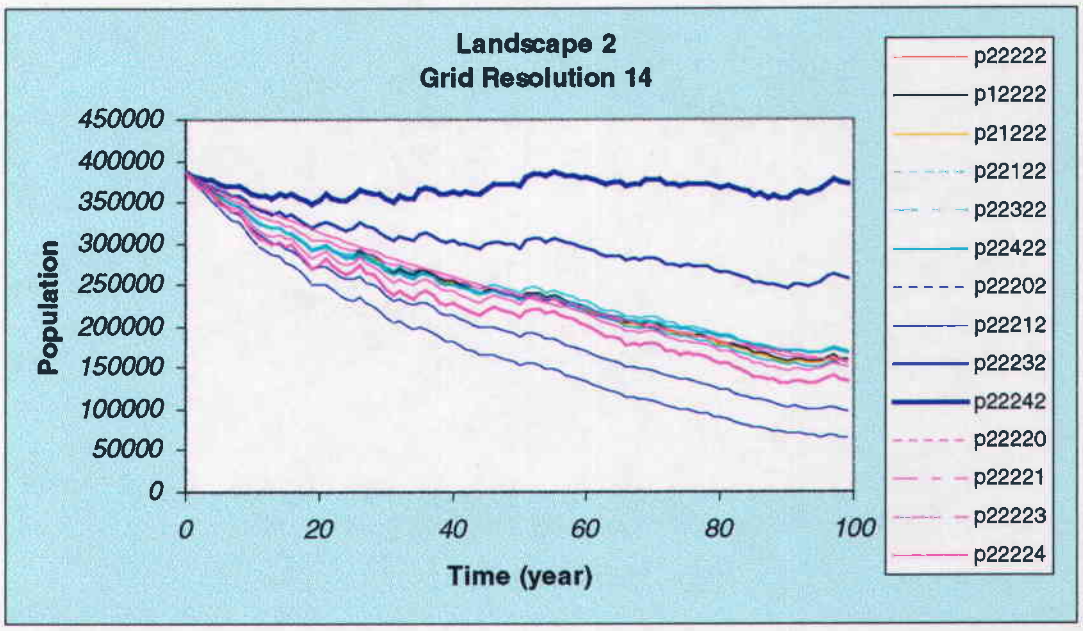


(a)

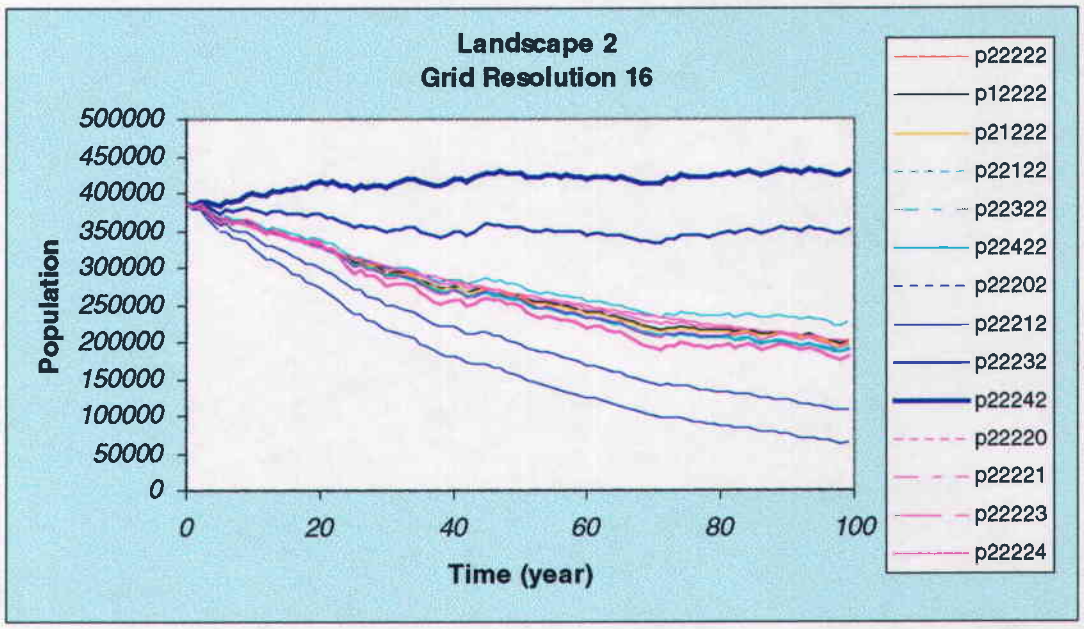


(b)

Figure 31. Projected population for landscape 1 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 2-2-2-2-2.

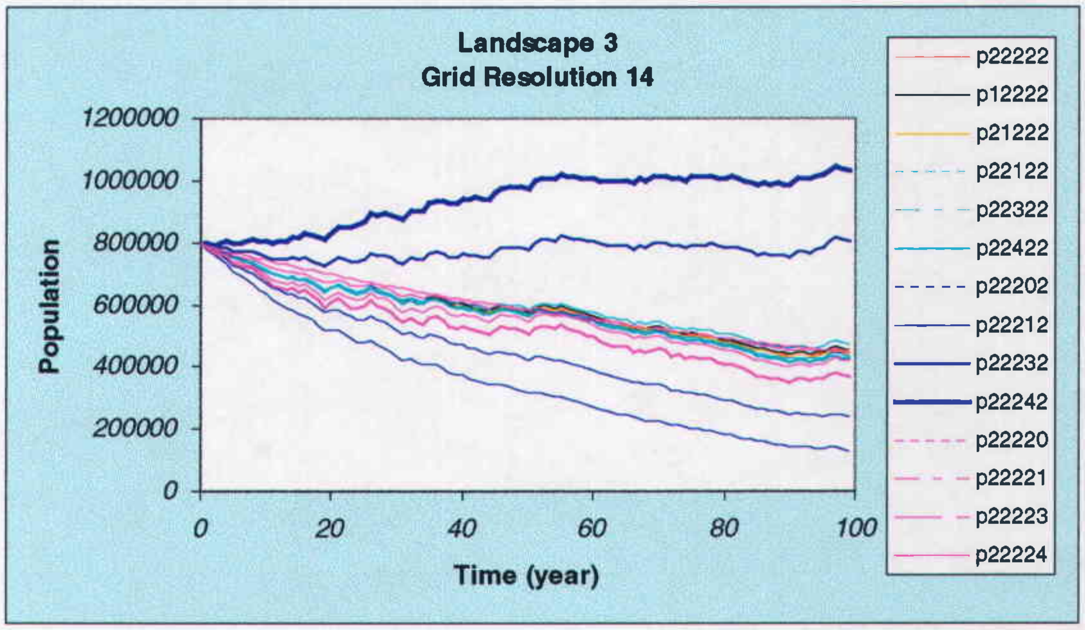


(a)

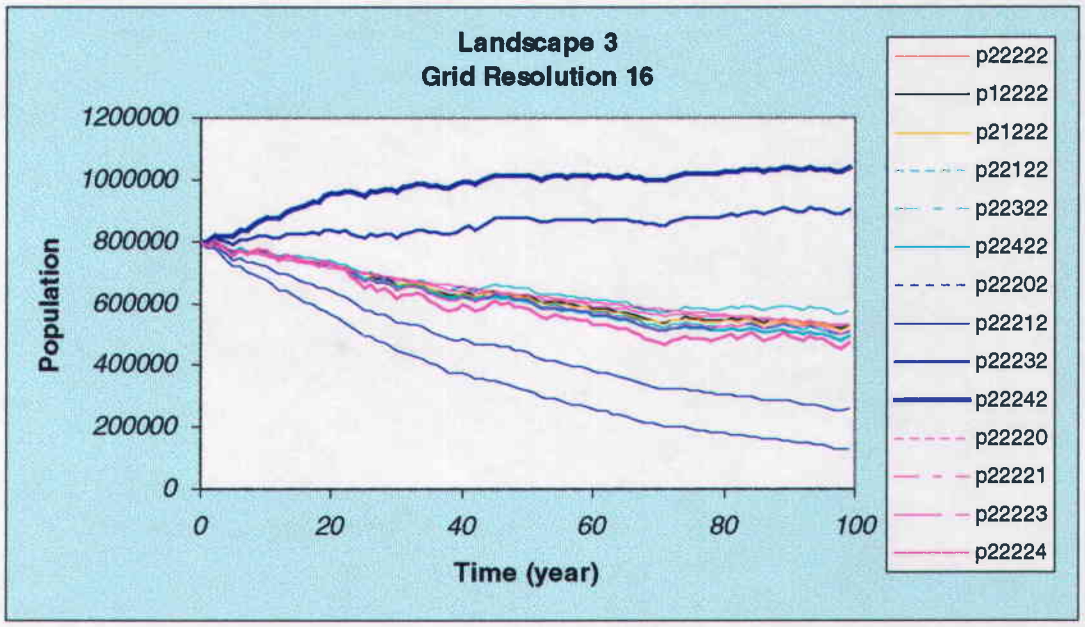


(b)

Figure 32. Projected population for landscape 2 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 2-2-2-2-2.

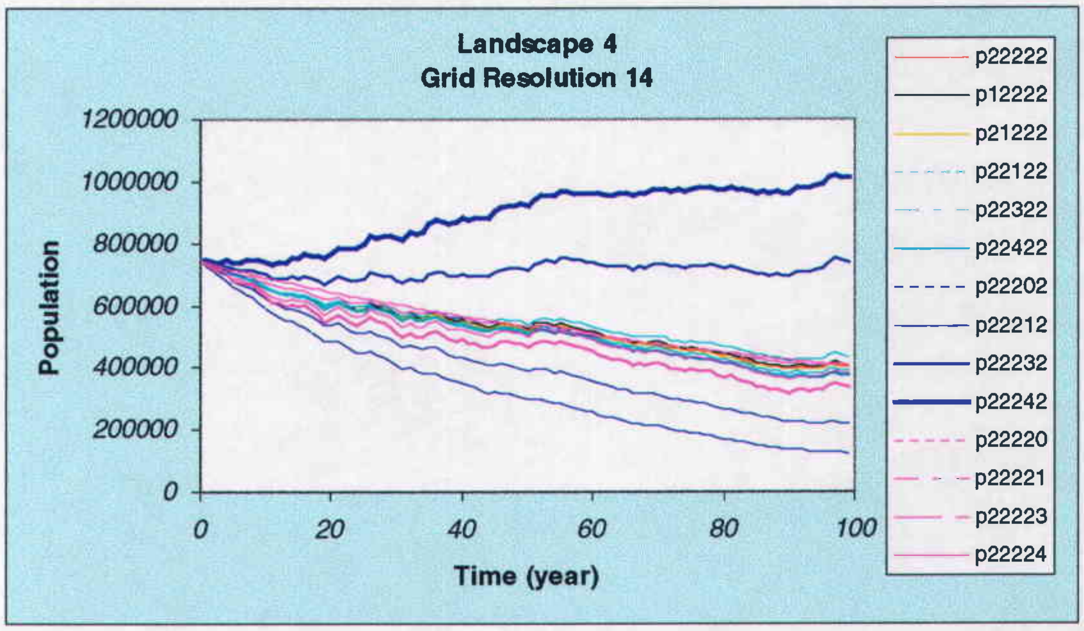


(a)

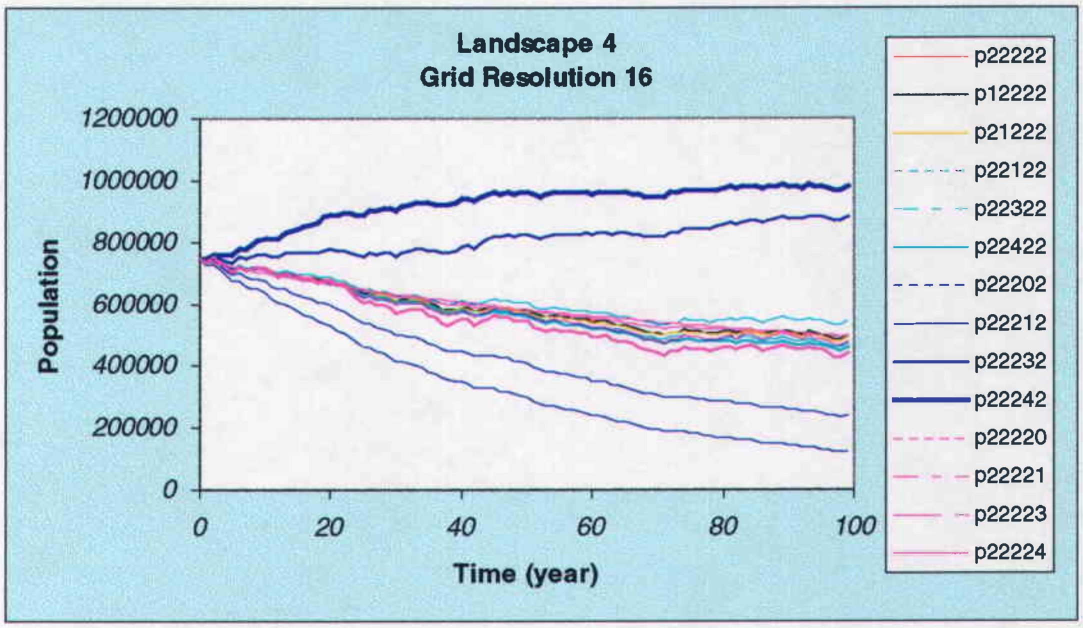


(b)

Figure 33. Projected population for landscape 3 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 2-2-2-2-2.



(a)



(b)

Figure 34. Projected population for landscape 4 in (a) grid resolution 14 and (b) grid resolution 16. Results are the responses to changes in model parameters with control parameter combination 2-2-2-2-2.

spatial rain distribution is the black line. Road effect (blue lines) is the most sensitive parameter in all landscapes and grid resolutions. With higher road effect, the population in landscape 1 increases while populations in three other landscapes decrease (the dotted blue line of no road effect is the highest line in landscape 1 and lowest lines in three other landscapes). This is an anticipated result, since only landscape 1 has a road value lower than the mean road value, while road values for the three other landscapes are higher than the mean road value.

The mobility of species and the effect of rain on survival and reproduction are the next most sensitive parameters. The highest effect of rain (bold magenta line) in most cases shows the lowest population relative to populations from different variations of that parameter. And higher mobility of the species shows lower population (bold aquamarine line is always lower than other lines of the same color). All other parameters, i.e. size-specific mobility (yellow line) and spatial distribution of rain (black color line), do not show any obvious sensitivity to model results.

3.4. Results from Elasticity Analysis

For the elasticity analysis, each matrix element is given a proportional change of positive 3 percent while all other elements remain constant. In the spatial model, elasticity analysis was conducted for landscape 1 and landscape 3 with parameter combinations 1-1-1-1-1 and 2-2-3-3-3 in grid resolution 14. Population growth rate lambda is calculated after 100-year simulation using formula : (population in year 100/ initial population)$^{(1/100)}$. The projection matrix has 17 non-zero elements. Elasticities, represented by lambda values in year 100, of those elements from simulations along with of the static projection matrix are summarized in Figure 35. The lambda values are the average from 20 replicate runs with twenty random seeds. The results show the same ranking of elasticities in all cases, both spatial or non-spatial. However, the values of lambda on the vertical axis are different between landscapes and parameter combinations. Landscape 1 with high road values always has the lower value of lambda relative to Landscape 3.

4. DISCUSSION

The results show that landscape 1 has a relative large amount of available resource, but suffers a great decline in population in comparisons with other landscapes. The obvious underlying cause of this decline is that the landscape has a high road density. High road density is a great threat to the tortoises, since the animals cannot recognize and avoid the danger of roads in the same pattern as they have ability to recognize the food gradient and move to the better habitat. The ability to move to the more favorable habitat is a natural behavior of the tortoise after thousands of years of evolution, while the impacts caused by roads have been present over the last few decades. The model indicated that sites with high road density suffer local population declines due to the killing effect of roads. Each year many tortoises in neighboring sites move to dangerous sites because of lower resource pressure, and continuously get killed by vehicles. By modeling road effects only, the results truly reflect the site-specific population dynamics caused by roads.

With rain effect only, the population dynamics in all landscapes show a decline in population, and a rather stable population growth rate lambda, which is almost the same as the lambda of the static projection matrix. This suggests that the fluctuations of natural factors, specifically rainfall in this study, are not a danger to tortoise populations, while human-related factors, or the road status in this study, may lead the population to extinction in a short time. Temporal patterns of rain cause the same effect to tortoise populations in all landscapes and fluctuate through time, while road effects are site-specific and remain constant through time. Therefore, local road status may be used as a guide to estimate site-specific population dynamics.

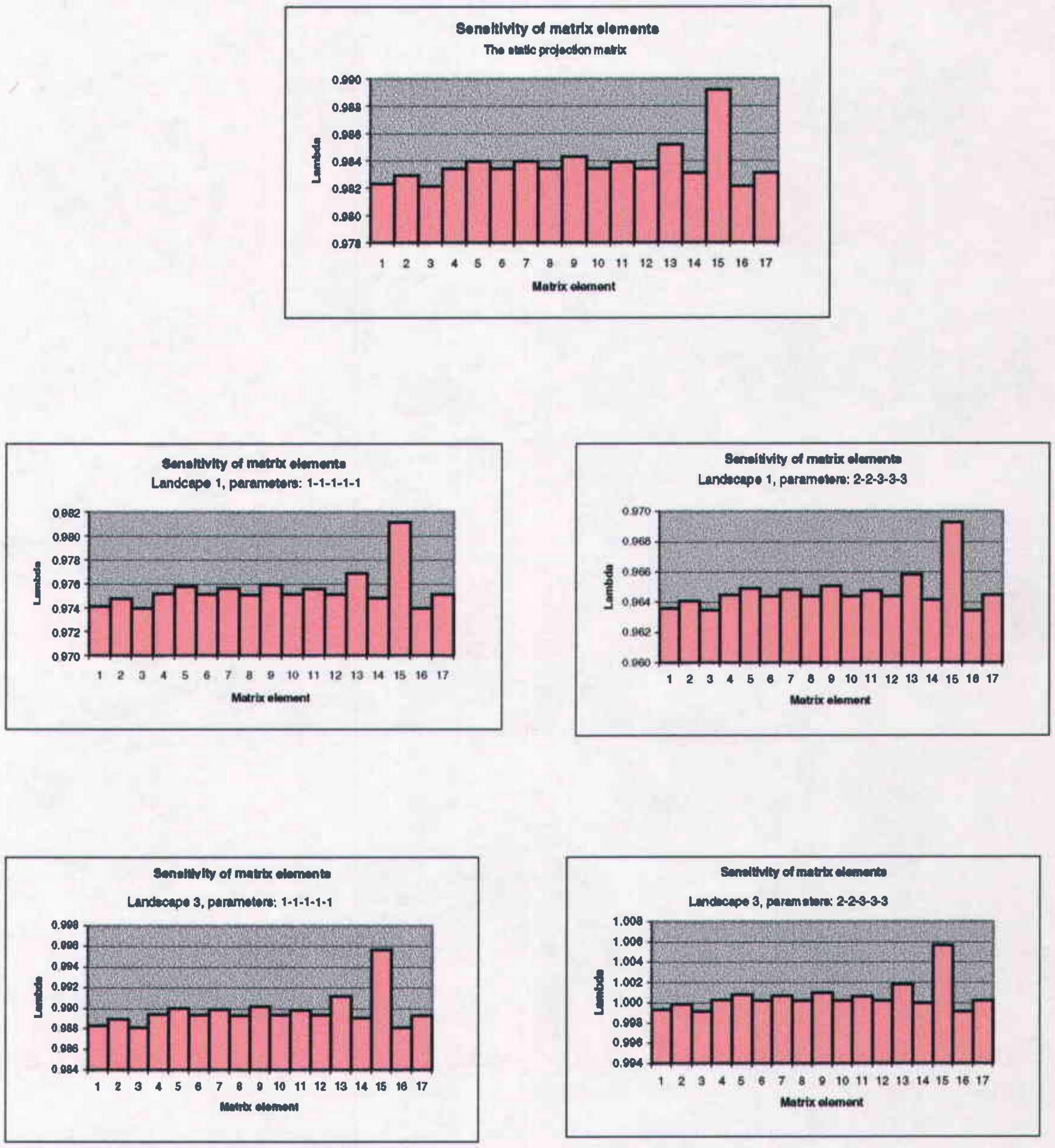


Figure 35. Results from elasticity analysis. Lambda values are calculated from 20 replicate simulations with an increase of 3 % of each matrix element at a time. Matrix elements are indexed from 1 to 17 with increment by row, from top to bottom, and left to right in the projection matrix (Table 1).

Rain effect, however, may become site-specific when rain patterns are looked at from both temporal and spatial contexts. Rain pattern in a site can be explained by the combined effects by temporal fluctuation of rain at region-scale, and topographic characteristics at local-scale. Analysis shows that 68-83 % of the variation in precipitation within a region can be explained by elevational differences in the Mojave Desert. In the Desert regions of California, the increase in precipitation is about 10 mm/100 m elevation. The residual variation is due to factors such as slope and aspect (Rowlands 1995). In this model, both temporal and spatial patterns of rain are modeled in a random process, so that it can more or less include the effects on tortoise population caused by the patchiness of rain within a landscape but fail to reflect the site-specific rain pattern. This should be taken into account in future studies.

Results from sensitivity analysis in all four sample landscapes indicate the same conclusion that roads are the most sensitive factor on the dynamics of the Desert tortoise's population. By using an unreliable model for road effects, the population in a site may be predicted to increase, while it decreases in reality. It is suggested that the effects of roads on the tortoise populations need to be considered for future research on the species. Rain effects also need to be taken into account, since the higher the effects of rain used in the model, the worse the status of population dynamics (the bold magenta line representing the highest effect of rain on survival and reproduction of the species is always under the other lines with the same color in Figures 27-34). These responses can be explained as follows: with high rain effect, both roads and rain can cause a drastic decline in local population in drought years, and the population can hardly recover even with the intervening years of high precipitation. Therefore, when spatial and temporal patterns of rain are well understood, the model can be used as a predictor for the population dynamics caused by variation of rainfall.

With regards to the sensitivity of mobility characteristics, mobility of species (aquamarine-teal lines) is more important than relative size-specific mobilities (yellow line). The more mobile the individuals of the species, the worse the situation in population dynamics (the bold teal line representing the highest effect of species mobility on movement is always equal to or under the other lines of the same color in Figures 27-

34). The possible reason is that the more mobile the individuals or the higher the number of animals moving out of a hexagon, the higher number of animals killed by vehicles. Once again, the dangerous effects of roads on the tortoise population are obvious.

Sensitivity of grid resolution shows interesting results. Population in the same landscape with the same parameter combination is higher at resolution 16 than at resolution 14 in both simulations conducted (Figure 26). The same results can be visualized also in Figures 27-34, the general trend of the population at resolution 16 is always better than at resolution 14. A possible explanation for this difference is that with finer resolution, animals can take the better advantages of resource availability in their occupying site, as well as in their neighborhood, by self-adjusting their movement. In the finer scale, landscapes become more homogeneous with respect to movement activities of the species. Research should be conducted to answer the questions 1) What is the most proper scale to model tortoise populations? and 2) Can a scale-independent model be constructed? 3) If not, how should the pattern-process relationships be modeled across scales? The biological and ecological characteristics of the species need to be studied carefully and scientifically in order to answer those questions.

Elasticities of demographic rates in the projection matrix in the spatial simulator have the same pattern as the static projection matrix. The only size-specific difference between the non-spatial and the spatial models is that the size-specific movement is included in the spatial simulator while this information is not included in the non-spatial matrix model. The same ranking of the importance of different vital rates in the two cases can by explained by the model structure that relative numbers of moving animals in different size classes are controlled more by the relative abundance of different size classes in the host hexagon than by the relative mobilities of those size classes. Therefore, the distribution of size classes in each hexagon almost always follows the stable size distribution, which is the right dominant eigenvector of the projection matrix used. As long as this situation is maintained, the elasticities will have the same pattern with elasticities of the static projection matrix. To increase the accuracy of the model, size-specific moving characteristics need to be taken into account in future research.

Study of elasticities of the mean population matrix needs to be analyzed carefully before applying to conservation. Change in one demographic rate can change the qualitative ranking of the elasticity values calculated from the projection matrix. Mills et al. (1999) found that for the desert tortoise case, changes across a realistic range of variation in the transition from class 6 to class 7 almost changed the ranking of the two highest elasticities. Additionally, when vital rates change to their high or low values observed in nature, prediction of future growth rate based on elasticities of the mean matrix can be misleading (Mills et al 1999). When its reliability is obtained, the spatial model constructed in this research can be an useful tool to study the actual changes of the population growth rate in responding to the changes in vital rate in nature and solve the above problems.

The upper respiratory tract disease (URTD) caused by the pathogen *Mycoplasma agassizii* is a causal factor in declines of desert tortoise populations in the California Mojave (Abbema, 1997). When the spatial distribution and the spreading pattern of the pathogen, and the effects of the disease on vital rates of the species are well understood, this spatial model can simulate the potential impact of the disease on tortoise's population dynamics.

The model has reached the initial step of constructing a reliable spatially explicit population model for the Desert tortoise. The results given by the model more or less have drawn the picture for the potential effects of different demographic and environmental factors on the tortoise's population dynamics. It is clear that the variations of spatial factors are important to explain the site-specific abundance of the tortoises. When spatial variations are taken into account, such as variation of roads in this study, the increases or decreases in local densities are independent, not as mentioned by Doak et al. (1994). A spatial model can be an useful tool to predict local population dynamics, especially when more GIS data become available and when future land-use scenarios for the area are spatially developed. In the future, field data need to be collected to construct more realistic relationships used in the model. The model then can be extended to include two-sex characteristics, demographic stochasticity, and the effects of other demographic and environmental factors.

BIBLIOGRAPHY

Abbema, J.V. (Editor). 1997. Proceedings: Conservation, Restoration, and Management of Tortoise and Turtles – An International Conference. New York Turtle and Tortoise Society and the WCS Turtle Recovery Program.

Berry, K.H. 1984. The Status of the Desert Tortoise (*Gopherus agassizii*) in the United States. Desert Tortoise Council Rept. U.S. Fish and Wildlife Service, Sacramento, California.

Bury, A.B. (Editor). 1982. North American tortoises: conservation and ecology. *Wildlife Research Report 12* of U.S. Department of the Interior Fish and Wildlife Service. Washington, D.C.

Bury, R.B. and Corn, P.S.1995. Have desert tortoises undergone a long-term decline in abundance?. *Wildlife Society Bulletin* 1995, 23 (1): 41-47

Caswell, H. 1989. Matrix population model - Construction, analysis, and interpretation. Sinauer Associates, Inc. Publishers. Sunderland, Massachusetts.

Doak, D., Kareiva, P., and Klepetka, B. 1994. Modeling population viability for the Desert Tortoise in the Western Mojave Desert. *Ecological Applications* 4 (3): 446-460

Dunning, J.B., Steward, D.J., Danielson, B.J., Noon, B.R., Root, T.L., Lamberson, R.H., and Stevens, E.E. 1995. Spatially explicit population models: current forms and future uses. *Ecological Applications* 5(1): 3-11.

Kiester, A.R., and Slatkin, M. 1974. A strategy of movement and resource utilization. *Theoretical Population Biology* 6: 1-20.

Kiester, A.R. 1985. Sex-specific dynamics of aggregation and dispersal in reptiles and amphibians. In: Rankin, M.L. (ed.) Migration: Mechanisms and adaptive significance. *Contrib. Marine Science* 27 suppl. Pp. 435-444.

Kimerling, A.J., White, D., Sahr, K., and Song, L. 1994. Mid-Project Report :Sampling design and statistics research for environmental monitoring and assessment program of development of global sampling grid. Terra Cognita: Oregon State University. Corvallis, OR. Technical Report 94-01.

Lefkovitch, L.P. 1965. The study of population growth in organisms grouped by stages. *Biometrics* 21: 1-18.

Leslie, P.H. 1945. On the use of matrices in certain population mathematics. *Biometrika* 33: 183-212.

Levin, S.A. 1992. The problem of pattern and scale in ecology. *Ecology* 73: 1943-1967

Lukenbach, R.A. 1982. Ecology and management if the Desert Tortoise (*Gopherus agassizii*) in California. In: Bury, A.B. (Editor). North American tortoises: conservation and ecology. *Wildlife Research Report 12* of U.S. Department of the Interior Fish and Wildlife Service. Washington, D.C.

Mills, L.S., Doak, F.D., and Wisdom, M.J. 1998. Reliability of conservation actions based on elasticities analysis of matrix models. Conservation Biology 13(4): 815-829.

Rowlands, P.G. 1995. *Regional Bioclimatology of the California Desert*. In: Latting, J. and Rowlands, P.G. (Editors). The California desert: An introduction to natural resources and man's impact. June Latting Books.

Sahr, K. 1998. Unpublished Directed research project. A system of the analysis of Discrete Grid Systems for multi-resolution location data.

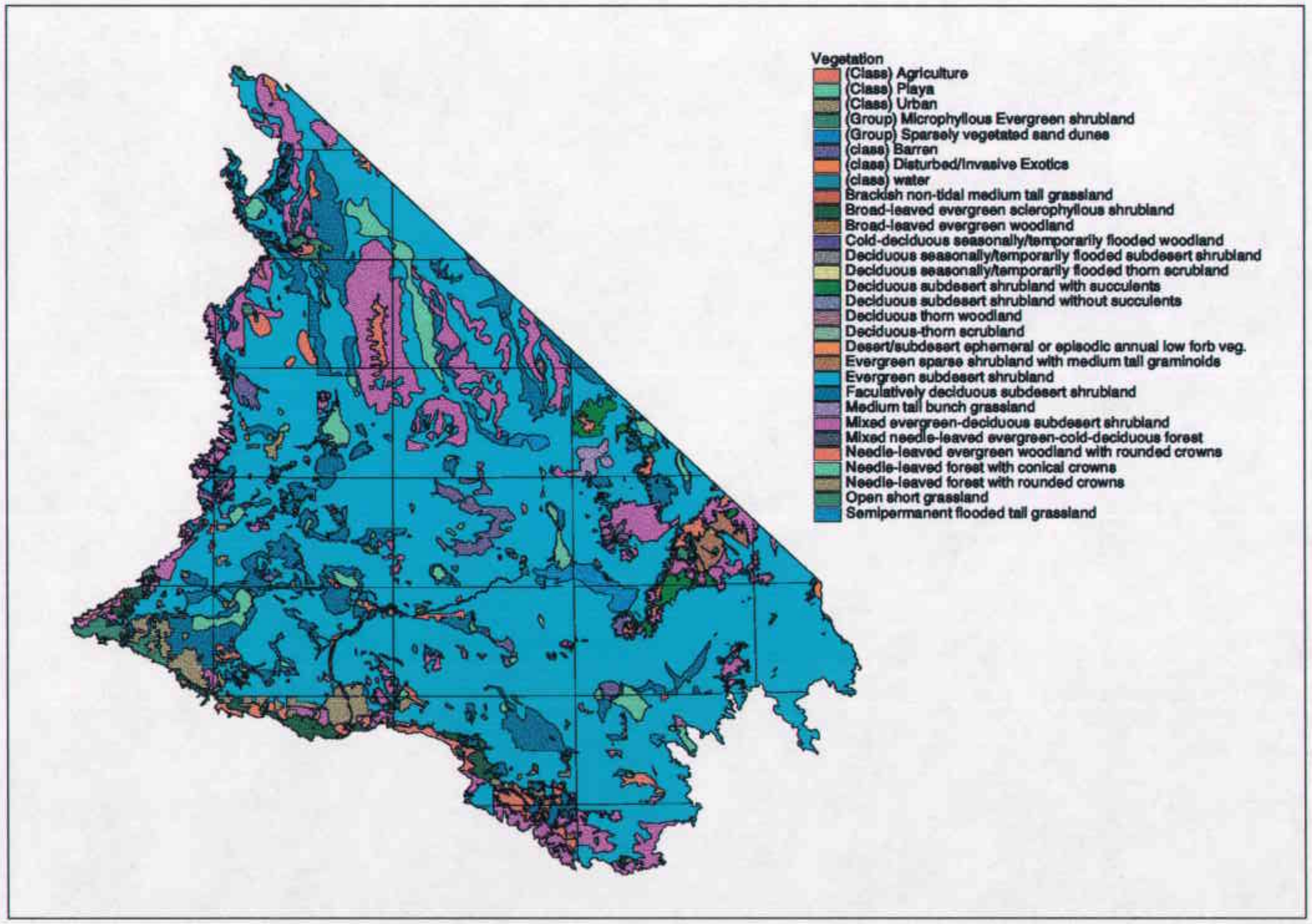
Schamberger, M.L., and Turner, F.B. 1986.The Application of habitat modeling to the Desert Tortoise (*Gopherus agassizii*). *Herpetologica* 42(1):134-138.

Schumaker, Nathan H. 1995. Dissertation. Habitat connectivity and Spotted Owl population dynamics. University of Washington.

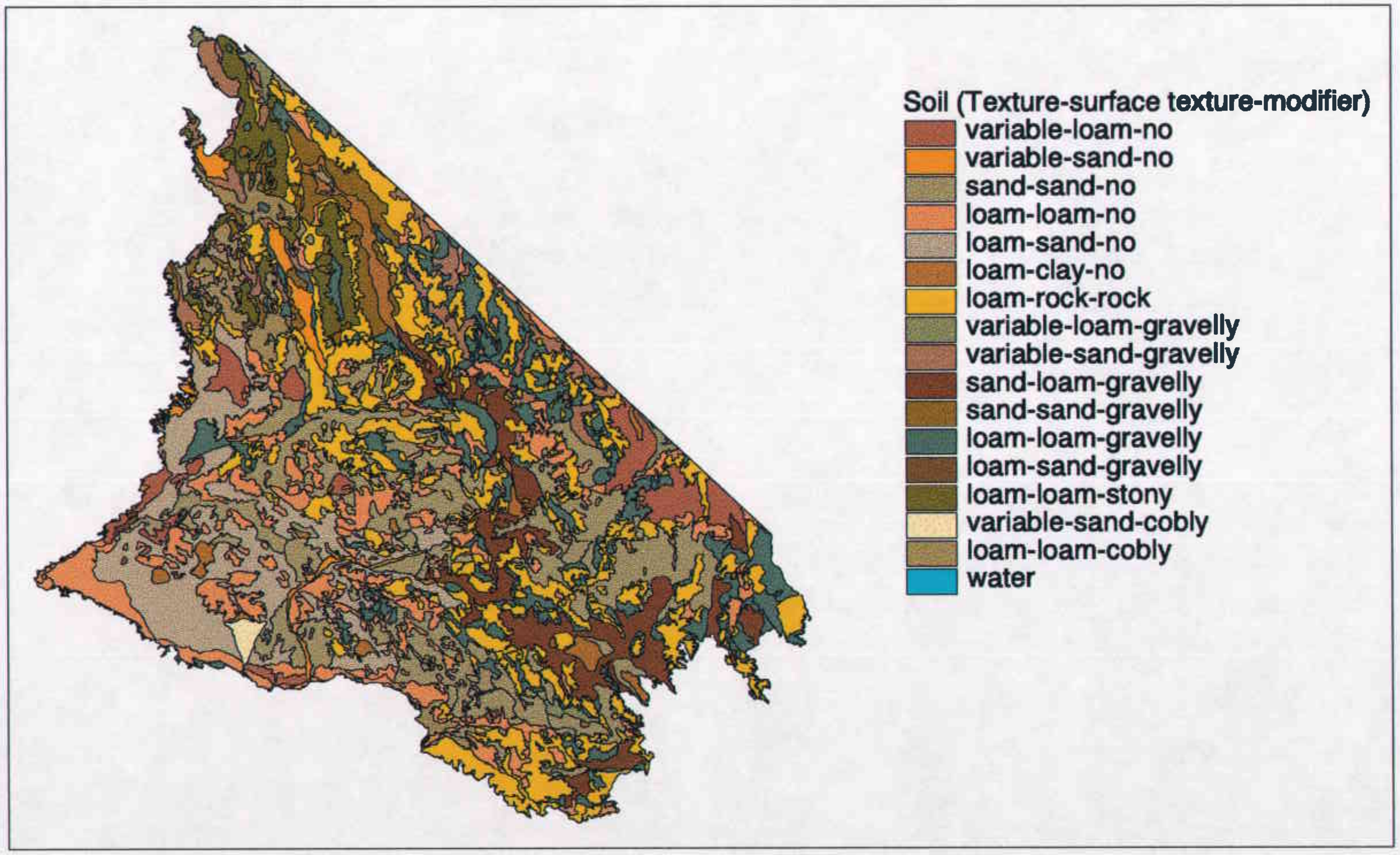
U.S. Fish and Wildlife Service 1994. The desert tortoise (Mojave population) recovery plan. U.S. Fish and Wildlife Service, Region 1 – Lead Region, Portland, Oregon.

APPENDICES

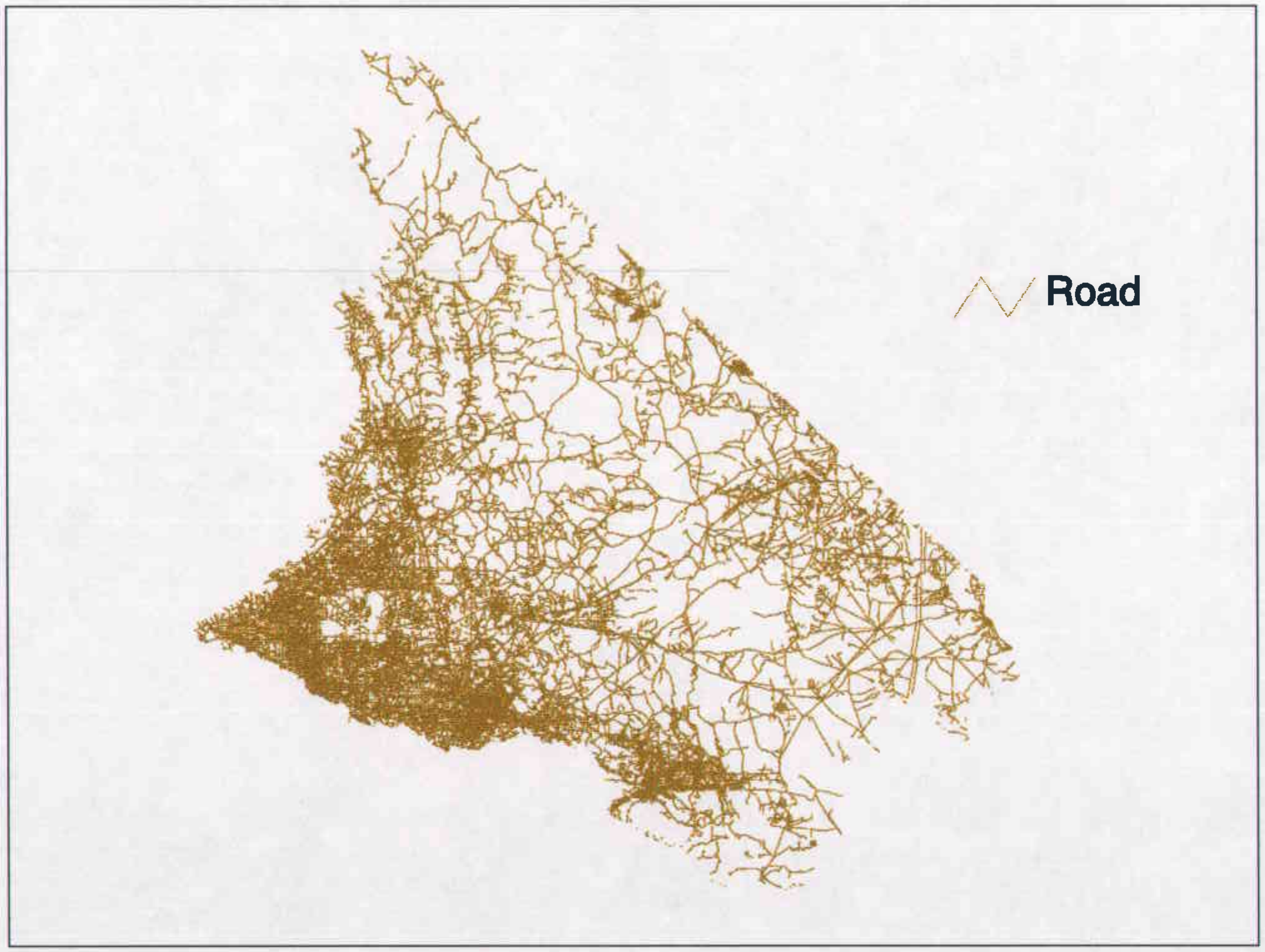
Apendix A. Arc/Info coverages used in the model


Vegetation
(Class) Agriculture
(Class) Playa
(Class) Urban
(Group) Microphyllous Evergreen shrubland
(Group) Sparsely vegetated sand dunes
(class) Barren
(class) Disturbed/Invasive Exotics
(class) water
Brackish non-tidal medium tall grassland
Broad-leaved evergreen sclerophyllous shrubland
Broad-leaved evergreen woodland
Cold-deciduous seasonally/temporarily flooded woodland
Deciduous seasonally/temporarily flooded subdesert shrubland
Deciduous seasonally/temporarily flooded thorn scrubland
Deciduous subdesert shrubland with succulents
Deciduous subdesert shrubland without succulents
Deciduous thorn woodland
Deciduous-thorn scrubland
Desert/subdesert ephemeral or episodic annual low forb veg.
Evergreen sparse shrubland with medium tall graminoids
Evergreen subdesert shrubland
Facultatively deciduous subdesert shrubland
Medium tall bunch grassland
Mixed evergreen-deciduous subdesert shrubland
Mixed needle-leaved evergreen-cold-deciduous forest
Needle-leaved evergreen woodland with rounded crowns
Needle-leaved forest with conical crowns
Needle-leaved forest with rounded crowns
Open short grassland
Semipermanent flooded tall grassland

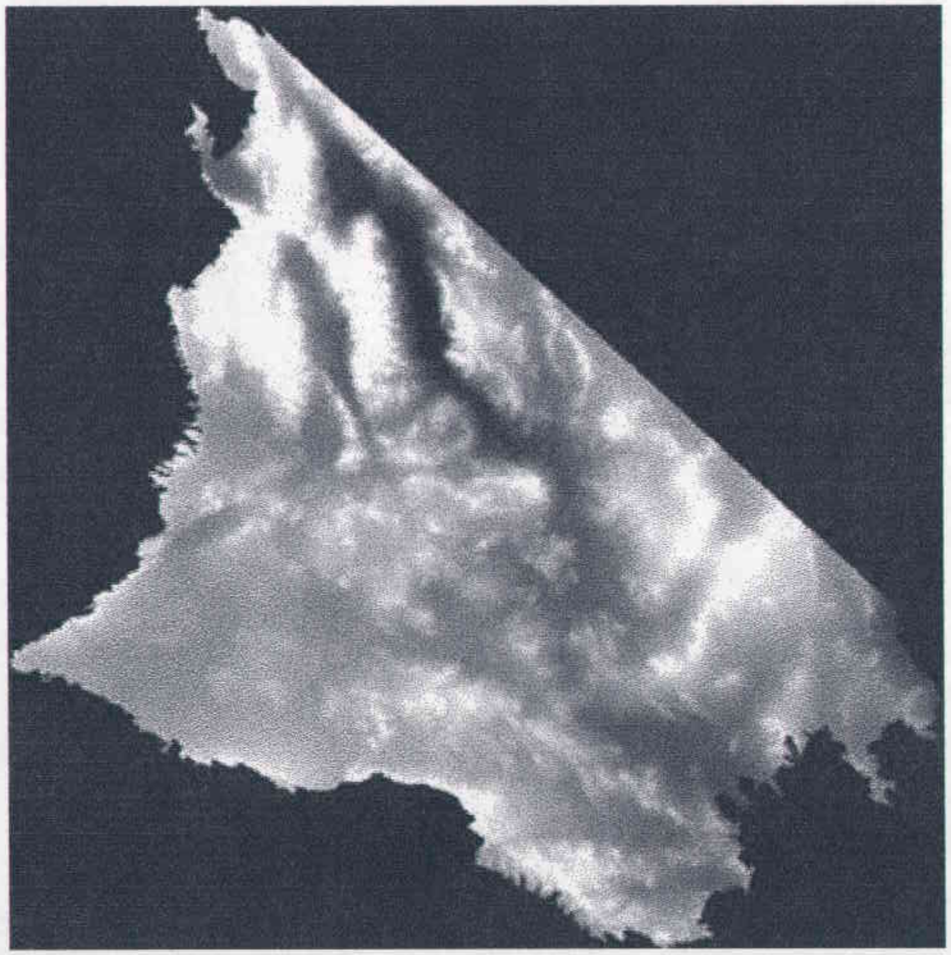
Vegetation map of the California Mojave Desert



Soil type map of the California Mojave Desert



Roads in the California Mojave Desert



Digital Elevation Model (DEM) of the California Mojave Desert

Appendix B. Explanation about model parameters

1. ***Grid resolution***

In this study, I use two grid resolutions 14 and 16 in Snyder projection, aperture 3. The area of each hexagon is 1.18759 sq.km. and 10.68832 Sq.km for resolution 14, and for resolution 16, respectively.

2. ***maxX*** and ***maxY***

maxX and ***maxY*** are the coordinates (i, j) of the upper-right corner of the hexagon grid. The values of maxX and maxY are 15 for grid resolution 14, and 47 for grid resolution 16. Therefore, each landscape consists of 16 rows and 16 columns in resolution 14, and of 48 rows and 48 columns in resolution 16.

3. ***stopTime***

This parameters is to specify the time step when the simulation will finish. This value is always set to 101, which means the simulation will be running in the time interval of 100 years.

4. ***resourceFile***

This is the name of the Ascii file containing the information about resource availability in each hexagon, which is used to initialize resource in each hexagon.

5. ***roadFile***

This is the name of the Ascii file containing the information about road value in each hexagon, which is used to initialize road value in each hexagon.

6. ***hexIdFile***

This is the name of the Ascii file containing the IDs of all hexagons in Snyder projection, which is used to specify the ID for each hexagon in the grid. The purpose of using this file is to write the outputs of each hexagon with its ID to further join the output information with the attribute table of the hexagon grid in ARC/Info for visualization.

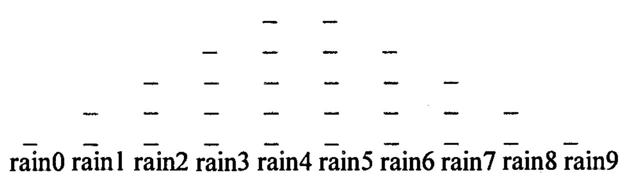
7. ***rainPattern*** This parameters has value of one or two. This is the first number in parameter combination used in the model. ***rainPattern*** is used to specify the distribution of rainfall values of all hexagons in the modeled landscape in each time step based on the mean rainfall (***meanRain***)of the whole area at that time step. The value of the variable ***meanRain*** at each time step is randomly sampled from a vector of 42-year annual rainfalls (1939-1980) recorded in Barstow Weather Station. After the ***meanRain*** is sampled, the next steps are:

- calculating the possible values of rainfall for each hexagon at the time step

    rain0 = 0.2 * ***meanRain***;
    rain1 = 0.4 * ***meanRain***;
    rain2 = 0.6 * ***meanRain***;
    rain3 = 0.8 * ***meanRain***;
    rain4 = 1.0 * ***meanRain;***
    rain5 = 1.0 * ***meanRain***;
    rain6 = 1.2 * ***meanRain***;
    rain7 = 1.4 * ***meanRain***;
    rain8 = 1.6 * ***meanRain***;
    rain9 = 1.8 * ***meanRain***;

- Specifying the distribution or the frequency histogram for each possible rainfall value.
This distribution reflects the level of patchiness of the rain in the modeled landscape in each single year

if (***rainPattern*** =1), the form of the distribution is



if (***rainPattern*** = 2), the form of the distribution is

```
 -      -     -     -     -     -     -     -     -     -
 -      -     -     -     -     -     -     -     -     -
 -      -     -     -     -     -     -     -     -     -

rain0 rain1 rain2 rain3 rain4 rain5 rain6 rain7 rain8 rain9
```

- Assigning a value of rainfall to each hexagon
At each time step, after the meanRain is sampled, the distribution above is constructed. Each distribution includes 30 values. The rainfall for each hexagon is then randomly sampled from that vector of 30 rainfall values. Therefore, the ***rainPattern*** 2 represents the higher deviation of the rainfalls in hexagons from the ***meanRain***, or the higher spatial variation of the rainfall in the area.
The purpose of specifying the distributions is to more or less include in the model the spatial variation of rainfall on the Desert Tortoise population dynamics.

8. ***mobilityPattern*** has values 1 or 2. This is the second number in parameter combinations used in the model.

Since the model tracks the number of moving animals in each size class in each time step, a vector of relative mobilities of all eight size classes needs to be specified. Specific values in the vector tell us that how mobile individuals in a size class are relative to individuals in other size classes. The idea, which is based on to set the values in the mobility vectors, is that the mobility of individuals increases from birth, and reaches the maximum when individuals are being juveniles, and then decreases when individuals are getting older and older until death. Two different vectors of mobilities are used in the model (see the code below):

```
if (mobilityPattern == 1) {
        mobilitySize0 = 1;
        mobilitySize1 = 2;
        mobilitySize2 = 3;
        mobilitySize3 = 4;
        mobilitySize4 = 4;
        mobilitySize5 = 3;
        mobilitySize6 = 2;
        mobilitySize7 = 1;
}

if (mobilityPattern == 2) {
        mobilitySize0 = 1;
        mobilitySize1 = 3;
        mobilitySize2 = 5;
        mobilitySize3 = 5;
        mobilitySize4 = 5;
        mobilitySize5 = 2;
        mobilitySize6 = 2;
        mobilitySize7 = 1;
}
```

9. ***MovingEqua*** has values 1, 2, 3, or 4. This is the third value in parameter combinations used in the model.
Following is the code to describe the role of this parameter:

```
        if (MovingEqua = = 1) {
         nMoving = .01 * varState().nIndividuals() *diff;}
    else if (MovingEqua = = 2) {
         nMoving = .03 * varState().nIndividuals() * diff;}
    else if (MovingEqua = = 3) {
         nMoving = .06 * varState().nIndividuals() * diff;}
```

```
else if (MovingEqua = = 4) {
     nMoving = .10 * varState().nIndividuals() * diff;}
else
     {exit(1);}
```

where

nMoving - the number of moving individuals out of the modeled hexagon at a time step
diff - the gradient of resource pressure (number of individuals per unit of resource) of the host hexagon and its best neighbor. This variable is discussed in detail in the **Model Structure** section above.
varState().nIndividuals() - This method returns the current number of individuals in the modeled hexagon

The reason for choosing this range is because it makes sure that the total number of moving animals out of a hexagon varies within a reasonable range with respect to the total number of individuals in that hexagon. If the constant is less than 0.01, the number of moving animals is too small, and if the constant is bigger than 0.10, the number of moving animals is too large, such as almost 100% of individuals in some hexagons. These four specific values are selected to test the sensitivity of the Moving Equation.

10. ***RepSurEquaRoad*** has values 0, 1, 2, 3, or 4. This is the forth number in parameter combinations used in the model.

This parameter is used to model the relationship between the road value in each site and the population dynamics on that site. The code below describes the specific values of this parameter:

```
if    (RepSurEquaRoad == 1) {
 roadFactor = 1.0 - 0.01 * (varState().nRoad()-MEAN_ROAD)/MEAN_ROAD;
}
else if (RepSurEquaRoad == 2) {
 roadFactor = 1.0 - 0.02 * (varState().nRoad() - MEAN_ROAD) / MEAN_ROAD;
}
else if (RepSurEquaRoad == 3) {
roadFactor = 1.0 - 0.03 * (varState().nRoad() - MEAN_ROAD) / MEAN_ROAD;
}
else if (RepSurEquaRoad == 4) {
 roadFactor = 1.0 - 0.04 * (varState().nRoad() - MEAN_ROAD) / MEAN_ROAD;
}

else {roadFactor = 1.0;}//So that 0 indicates no road effect taken into account.
```

varState().nRoads() - the method returns the road value in the modeled hexagon
The general form of theses equations is discussed in detail in the **Model Structure** section above.
The constants 0.01, 0.02, 0.03, and 0.04 are chosen to make sure that the roadFactor varies in a reasonable range with respects to the variations of all elements in the corresponding projection matrix.

11. ***RepSurEquaRain*** has values 0, 1, 2, 3, or 4. This is the fifth number in parameter combinations used in the model.

This parameter is used to model the relationship between the rainfall in each site and the population dynamics on that site. The code below describes the specific values of this parameter:

```
if (RepSurEquaRain = = 1) {
     rainFactor = 1.0 + (0.01 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
}

else if (RepSurEquaRain = = 2) {
     rainFactor = 1.0 + (0.02 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
}

else if (RepSurEquaRain = = 3) {
     rainFactor = 1.0 + (0.03 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
}
```

```
else if (RepSurEquaRain = = 4) {
      rainFactor = 1.0 + (0.04 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
}

else {rainFactor = 1.0;}//So that 0 indicates no rain effect taken into account.
```

Where ***rainCellvalue*** is randomly sampled for each hexagon from the vector of 30 rainfall values in the instant time step, as discussed in the section above about the parameter ***rainPattern***.
The general form of theses equations is discussed in detail in the **Model Structure** section above.
The constants 0.01, 0.02, 0.03, and 0.04 are chosen to make sure that the variable *rainFactor* varies in a reasonable range with respects to the variations of all elements in the corresponding projection matrix.

12. ***RandSeed***

This parameter allows to set the random seed to start the random number generator in the simulation.

13. ***gridResolution*** has values 14 or 16

14. ***useGraphics*** has values 0 or 1.

0 means no graphic visualization is desired, and 1 means vice versa.

15. ***useOutput*** --> 0 or 1 (set 1 for writing to files)

0 means writing model's results to files is desired, and 1 means vice versa.

16. ***outFileName***

This is the name of the output file. This output file contains the information about population, such as total population, population growth rate, of the whole modeled landscape in each time step.

17. ***outputCellFreq***

This parameter is used to specify the frequency (number of time-steps) for writing cell-specific information to files.

18. **pm#** has values 1, 2, ..., 17

This parameter is the index of the element in the static projection matrix that will be changed in order to calculate elasticity.

19. **percentChange**

This parameter is used to specify the percent change of the matrix element specified in the previous step. Values of percentChange can be negative or positive, but vary within the range from minus 5 to plus 5 is desired to make sure that all elements of the matrix vary in reasonable ranges, i.e. the survival rates of all size classes are always less than one.

Apendix C. C++ codes for construction and analysis of the model

```
/////////////////////////////////////////////////////////////////////////////
//
// DgResourceCell.h
//This is the header file for building cell's structures and methods
//
/////////////////////////////////////////////////////////////////////////////

#ifndef DGRESOURCECELL_H
#define DGRESOURCECELL_H
#include <iostream>
using namespace std;
#include "DgSimCell2DS.h"
#include "TcColor.h"
#include "fstream.h";
class DgResourceSim;

/////////////////////////////////////////////////////////////////////////////
class DgCountRes  {

   public:

      DgCountRes (double nIndividualsIn =0.0, double nResourceIn = 0.0,
double nRoadIn = 0.0, int hexIdIn = 0, double rainCellIn = 0.0,
double nSize0In = 0.0, double nSize1In =0.0, double nSize2In= 0.0,
double nSize3In =0.0, double nSize4In = 0.0, double nSize5In= 0.0, double
nSize6In = 0.0, double nSize7In = 0.0)
           : nIndividuals_ (nIndividualsIn), nResource_ (nResourceIn),
nRoad_(nRoadIn), hexId_(hexIdIn), rainCell_(rainCellIn),
nSize0_(nSize0In), nSize1_(nSize1In), nSize2_(nSize2In), nSize3_(nSize3In),
nSize4_(nSize4In), nSize5_(nSize5In), nSize6_(nSize6In), nSize7_(nSize7In) { }

      DgCountRes (const DgCountRes& countRes)
           : nIndividuals_ (countRes.nIndividuals()),
             nResource_  (countRes.nResource()),
             nRoad_(countRes.nRoad()),
             hexId_(countRes.hexId()),
             rainCell_(countRes.rainCell()),
             nSize0_(countRes.nSize0()),
             nSize1_(countRes.nSize1()),
             nSize2_(countRes.nSize2()),
             nSize3_(countRes.nSize3()),
             nSize4_(countRes.nSize4()),
             nSize5_(countRes.nSize5()),
             nSize6_(countRes.nSize6()),
             nSize7_(countRes.nSize7()) { }

      DgCountRes& operator= (const DgCountRes& cRes)
      { if (&cRes != this)
         nIndividuals_ = cRes.nIndividuals();
         nResource_ = cRes.nResource();
           nRoad_ = cRes.nRoad();
           hexId_  = cRes.hexId();
           rainCell_ = cRes.rainCell();
         nSize0_ =cRes.nSize0();
           nSize1_ = cRes.nSize1();
           nSize2_ = cRes.nSize2();
           nSize3_ = cRes.nSize3();
           nSize4_ = cRes.nSize4();
           nSize5_ = cRes.nSize5();
           nSize6_ = cRes.nSize6();
         nSize7_ = cRes.nSize7();
        return *this; }

      bool operator== (const DgCountRes& cRes)
           { return (cRes.nIndividuals() == nIndividuals() &&
                 cRes.nResource() == nResource() &&
                 cRes.nRoad()  == nRoad()  &&
                     cRes.hexId()  == hexId()  &&
                     cRes.rainCell() == rainCell() &&
                     cRes.nSize0() == nSize0() &&
                     cRes.nSize1() == nSize1() &&
                     cRes.nSize2() == nSize2() &&
                     cRes.nSize3() == nSize3() &&
                     cRes.nSize4() == nSize4() &&
                     cRes.nSize5() == nSize5() &&
                     cRes.nSize6() == nSize6() &&
                    cRes.nSize7() == nSize7() );}

      bool operator!= (const DgCountRes& cRes) { return !operator==(cRes); }

      string asString (void) const { return string(dbl2str(nIndividuals()) + "|"+
                                         dbl2str(nResource())); }

      void setNIndividuals (double nIndividualsIn) { nIndividuals_ = nIndividualsIn;}
      void setNResource (double nResourceIn) { nResource_ = nResourceIn; }
      void setNRoad  (double nRoadIn)  {nRoad_  = nRoadIn; }
      void setRainCell (double rainCellIn) {rainCell_ = rainCellIn;}
      void setHexId  (int hexIdIn)     {hexId_    = hexIdIn;}
      void setNSize0 (double nSize0In) {nSize0_ = nSize0In;}
      void setNSize1 (double nSize1In) {nSize1_ = nSize1In;}
      void setNSize2 (double nSize2In) {nSize2_ = nSize2In;}
      void setNSize3 (double nSize3In) {nSize3_ = nSize3In;}
      void setNSize4 (double nSize4In) {nSize4_ = nSize4In;}
      void setNSize5 (double nSize5In) {nSize5_ = nSize5In;}
      void setNSize6 (double nSize6In) {nSize6_ = nSize6In;}
      void setNSize7 (double nSize7In) {nSize7_ = nSize7In;}

      double nIndividuals (void) const { return nIndividuals_; }
      double nResource (void) const { return nResource_; }
      double nRoad  (void) const {return nRoad_; }
      double rainCell (void) const {return rainCell_;}
      int hexId (void) const {return hexId_;}
      double nSize0 (void) const {return nSize0_;}
      double nSize1 (void) const {return nSize1_;}
      double nSize2 (void) const {return nSize2_;}
      double nSize3 (void) const {return nSize3_;}
      double nSize4 (void) const {return nSize4_;}
      double nSize5 (void) const {return nSize5_;}
      double nSize6 (void) const {return nSize6_;}
      double nSize7 (void) const {return nSize7_;}



  protected:

      double nIndividuals_;
      double nResource_;
      double nRoad_;
      double rainCell_;
      int hexId_;
      double nSize0_;
      double nSize1_;
      double nSize2_;
      double nSize3_;
      double nSize4_;
      double nSize5_;
      double nSize6_;
      double nSize7_;

};

////////////////////////////////////////////////////////////////////////////////
inline ostream& operator<< (ostream& stream, const DgCountRes& cRes)
{
   return stream << cRes.asString();

} // ostream& operator<<

////////////////////////////////////////////////////////////////////////////////
class DgResourceCell :  public DgSimCell2DS<void*, DgCountRes, DgResourceCell> {

   public:

      DgResourceCell (void) : resourceSim_ (0) { }

      DgResourceCell (DgResourceSim& dbIn);

      DgResourceSim* resourceSim (void) { return resourceSim_; }
      DgResourceCell& operator= (const DgResourceCell& cell)
         { DgSimCell2DS<void*, DgCountRes, DgResourceCell>::operator=(cell);
           return *this; }

      virtual string asString (void) const { return varState().asString(); }
      virtual int initialize (const DgLocation& loc); // called once before sim runs
      virtual int reset (void);                         // called before each sim run
      virtual int process (void);                       // called each iteration
      virtual int setGraphicState (void);               // called before cells are drawn
      virtual int setOutputState (void);                // called before cells are output
      virtual int postProcess (void) { return 0; }    // called after sim runs

   protected:

      DgResourceSim* resourceSim_;

};

////////////////////////////////////////////////////////////////////////////////
inline ostream& operator<< (ostream& stream, const DgResourceCell& cell)
{
   return stream << cell.location() << ":" << cell.varState();

} // ostream& operator<<

////////////////////////////////////////////////////////////////////////////////
//
//End of DgResourceCell.h
//
////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////
//
// DgResourceSim.h
// This is the header file containing structures and methods for the simulator
// and other classes used in programming the model.
//
/////////////////////////////////////////////////////////////////////////////////////////

#ifndef DGRESOURCESIM_H
#define DGRESOURCESIM_H
#define ROW 8
#define COL 8
#define MAX_RECORD 4000
#define MEAN_RAIN 402.0238 //This value is the mean of annual rainfalls in 42-year period
#define MEAN_ROAD 5.652 //This value is the mean of road values in four sample landscapes

#include <iostream>

using namespace std;
#include <stdlib.h>
#include "DgSimDB2DS.h"
#include "DgResourceCell.h"


//////////////// Declare the non-zero values for the static projection matrix

double pm05 = 2.22;
double pm06 = 3.38;
double pm07 = 4.38;
double pm10 = 0.716;
double pm11 = 0.567;
double pm21 = 0.149;
double pm22 = 0.567;
double pm32 = 0.149;
double pm33 = 0.604;
double pm43 = 0.235;
double pm44 = 0.560;
double pm54 = 0.225;
double pm55 = 0.678;
double pm65 = 0.249;
double pm66 = 0.851;
double pm76 = 0.016;
double pm77 = 0.860;
```

```
///////////////////The structure of the class matrix:

class matrix {
protected:
        int m;
        int n;
        double a[ROW][COL];
public:
       matrix (void){
                m = ROW;
                n = COL;
                for (int i =0; i<m; i++) {
                        for (int j = 0; j<n; j++)
                                a[i][j] = 0.0;
                }
        }

        matrix (int ml, int nl){
                m = ml;
                n = nl;
                for (int i=0; i<m; i++) {
                        for (int j=0; j<n; j++)
                                a [i][j] = 0.0;
                }
        }

        /////////


        matrix & operator = (const matrix & matr) {
                if (&matr != this)
                m = matr.m;
                n = matr.n;
                for (int i =0; i<m; i++) {
                        for (int j =0; j<n; j++)
                                a[i][j] = matr.a[i][j];
                }
                return *this;
        }

        ////////

        matrix operator * (const matrix matr) {
                if (n != matr.m) {
                        cout <<"This 2 matrices can't be multiplied.";
                        return *this;
                }
                else
                {
                        matrix product (m, matr.n);
                        for (int i=0; i<m; i++){
                                for (int j=0; j<matr.n; j++){
                                        double sum =0;
                                        for (int k =0; k<n;k++){
                                                sum =sum + a[i][k]*(matr.a[k][j]);
                                        }
                                        product.a[i][j] = sum;
                                }
                        }
                        return product;
                }
        }

        ////////////

        matrix operator * (double doub) {
                for (int i =0; i<8;i++) {
                        for ( int j = 0; j < 8; j++ ) {
                                a[i][j] = a[i][j]*doub;
                        }
                }
                return *this;
        }
////////////

        void print() {
                cout<< "Matrix ["<<m<<"]["<<n<<"]\n";
                for (int i=0; i<m; i++){
                        for (int j=0; j<n;j++) {
                                cout.width(10);
                                cout <<a[i][j];
                        }
                        cout <<"\n";
                }
        }


/////////////
        void replace (double a0, double a1, double a2, double a3, double a4, double a5, double
a6, double a7)
      {
                m = 8;
                n = 1;
                a[0][0] = a0;
                a[1][0] = a1;
                a[2][0] = a2;
                a[3][0] = a3;
                a[4][0] = a4;
                a[5][0] = a5;
                a[6][0] = a6;
                a[7][0] = a7;
        }

/////////////

        double get_value (int p, int q) {
                return a[p][q];
        }
        void projection_matrix (){
                a[0][5] = pm05;
                a[0][6] = pm06;
                a[0][7] = pm07;
                a[1][0] = pm10;
                a[1][1] = pm11;
                a[2][1] = pm21;
                a[2][2] = pm22;
                a[3][2] = pm32;
                a[3][3] = pm33;
                a[4][3] = pm43;
                a[4][4] = pm44;
                a[5][4] = pm54;
                a[5][5] = pm55;
                a[6][5] = pm65;
                a[6][6] = pm66;
                a[7][6] = pm76;
                a[7][7] = pm77;
        }

};

////////////The structure of the class Vector:
class Vector {

protected:
        int size;
        double a[MAX_RECORD];

public:
        Vector (void) {
                size = MAX_RECORD;
                for (int i = 0; i < MAX_RECORD; i++ )

                        a[i] = 0.0;

        }

        Vector (int size1){
                size = size1;
                for (int i =0; i<size1; i++)
                        a[i] = 0.0;
        }
        Vector & operator = (const Vector & vec) {
                size = vec.size;
                for (int i =0; i< vec.size; i++)

                        a[i] = vec.a[i];

                return *this;

        }

        void print () {

                cout << "\n VECTOR : \n";

                for (int i =0; i< size; i++)

                        cout <<"["<<i<<"]  " <<a[i] <<"\n";

        }


        double get_value (int i) {

                return a[i];

        }

        void put_value (int i, double doub) {

                a[i] = doub;

        }

        friend void readfile_to_vec (Vector &, char *);
        friend double randinVector (Vector);

};

void readfile_to_vec (Vector & vec, char* filename) {

      ifstream fin;
        char str[10];
        double str_atof;
        int i = 0;
      char ch;
      int j = 0;
      fin.open (filename, ios::in);
        if (!fin.is_open()) {
        cout <<"Can't open file input to read.\n";
        exit(1);
        }
      while ((fin.eof() != 1) || (j == vec.size-1)) {
                        fin.get(ch);
                        if (ch==' ' || ch=='\t' || ch == '\n') {
                                if (i==0) continue;
                                str[i] = '\0';
                                str_atof = atof (str);
                                vec.put_value (j, str_atof);
                                i=0;
                                j++;
                                continue;
                        }
                        str[i] = ch;
                        i++;
        }
        if (i!=0) {
                str[i] = 0;
                str_atof = atof (str);
                vec.put_value (j, str_atof);
        }

        fin.close();
}

double randinVector (Vector vec) {
      double x;
      int j;
        int i = RAND_MAX / vec.size;
        x = rand();
        j = x / i;
      return vec.get_value (j);
}
```

////////////The structure of class DgResourceSim:

```
class DgResourceSim :  public DgSimDB2DS<DgResourceCell> {

   public:

      DgResourceSim (DgPhysicalRFS2D<DgResourceCell>& rfIn,
                     double indiv = 10.0, double size0 = 10.0, double size1 = 10.0,
                     double size2 = 10.0, double size3 = 10.0, double size4 = 10.0,
                     double size5 = 10.0, double size6 = 10.0, double size7 = 10.0,
                         bool useGraphicsIn = true, bool useOutputIn = false,
                         const string& outFileNameIn = "output" );


      virtual int initialize  (void);
      virtual int reset       (void);
      virtual void output     (void);
      virtual int setOutputState (void);

      double nIndiv (void) const { return nIndiv_; }
      double nSize0Indiv (void) const {return nSize0Indiv_;}
      double nSize1Indiv (void) const {return nSize1Indiv_;}
      double nSize2Indiv (void) const {return nSize2Indiv_;}
      double nSize3Indiv (void) const {return nSize3Indiv_;}
      double nSize4Indiv (void) const {return nSize4Indiv_;}
      double nSize5Indiv (void) const {return nSize5Indiv_;}
      double nSize6Indiv (void) const {return nSize6Indiv_;}
      double nSize7Indiv (void) const {return nSize7Indiv_;}

      void setNIndiv (double nIndivIn) { nIndiv_ = nIndivIn; }
      void setNSize0Indiv (double nSize0IndivIn) {nSize0Indiv_ = nSize0IndivIn;}
      void setNSize1Indiv (double nSize1IndivIn) {nSize1Indiv_ = nSize1IndivIn;}
      void setNSize2Indiv (double nSize2IndivIn) {nSize2Indiv_ = nSize2IndivIn;}
      void setNSize3Indiv (double nSize3IndivIn) {nSize3Indiv_ = nSize3IndivIn;}
      void setNSize4Indiv (double nSize4IndivIn) {nSize4Indiv_ = nSize4IndivIn;}
      void setNSize5Indiv (double nSize5IndivIn) {nSize5Indiv_ = nSize5IndivIn;}
      void setNSize6Indiv (double nSize6IndivIn) {nSize6Indiv_ = nSize6IndivIn;}
      void setNSize7Indiv (double nSize7IndivIn) {nSize7Indiv_ = nSize7IndivIn;}

      const vector<TcColor*>& cols (void) const { return cols_; }

   private:

      double nIndiv_;
      double nSize0Indiv_;
      double nSize1Indiv_;
      double nSize2Indiv_;
      double nSize3Indiv_;
      double nSize4Indiv_;
      double nSize5Indiv_;
      double nSize6Indiv_;
      double nSize7Indiv_;
      vector<TcColor*> cols_;

   friend class DgResourceCell;

};

int nHex = 3600;
int hexNumber;

Vector rainVec (42); //Rainfall data will be read into this vector(see reset() in
DgResourceSim.C)

Vector rainCellVector (30); //The rainfall for each hexagon is randomly sampled from this
Vector.
                                                 //This Vector is changed every year (See DgResourceCell.C).
```

```
matrix Leslie (8,8); //Projection matrix will be read into this matrix (see reset() in
DgResourceSim.C).

double maxRes = 100;

double mobilitySize0 = 1;
double mobilitySize1 = 2;
double mobilitySize2 = 3;
double mobilitySize3 = 4;
double mobilitySize4 = 4;
double mobilitySize5 = 3;
double mobilitySize6 = 2;
double mobilitySize7 = 1;

char out_string[100];
char outCell_string[100];

int StopTime = 0;

double totalSize0 = 0;
double totalSize1 = 0;
double totalSize2 = 0;
double totalSize3 = 0;
double totalSize4 = 0;
double totalSize5 = 0;
double totalSize6 = 0;
double totalSize7 = 0;

double lambda;
double totalPopulation = 0;
double totalPopLastYear;


char* resFileName;
char* roadFileName;
char* hexIdFileName;
char* outCellFileName;
int outCellFreq = 10;
ofstream fout;

int rainPattern = 1;
int mobilityPattern = 1;
int MovingEqua = 1;
int RepSurEquaRoad = 1;
int RepSurEquaRain = 1;
int randSeed = 0;
int gridResolution = 14;

//////////////////////////////////////////////////////////////////////////////////
//
//End of DgResourceSim.h:
//
///////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////
//
// DgResourceCell.C
// This file is for building structure and methods for each simulation cell
//
///////////////////////////////////////////////////////////////////////////////////

#include <iostream>

using namespace std;

#include "DgResourceCell.h"
#include "DgResourceSim.h"
#include "DgSimDB2DS.h"
#include <fstream.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
```

```
////////////////////////////////////////////////////////////////////////////////
DgResourceCell::DgResourceCell (DgResourceSim& dbIn)
   : DgSimCell2DS<void*, DgCountRes, DgResourceCell>(dbIn), resourceSim_ (&dbIn)
{

} // DgResourceCell::DgResourceCell

////////////////////////////////////////////////////////////////////////////////
int
DgResourceCell::initialize (const DgLocation& loc)
//
// call once before simulation runs
//
{
   if (DgSimCell2DS<void*, DgCountRes, DgResourceCell>::initialize(loc))
   {
      return -1;
   }

   setGraphicState();
   setDrawBoundary(true);
   setDrawPoint(false);
   setDrawLabel(false);

   return 0;

} // DgResourceCell::initialize

////////////////////////////////////////////////////////////////////////////////
int
DgResourceCell::reset (void)

//
// call before each simulation run
//
{
   setGraphicState();

   return 0;

} // DgResourceCell::reset

////////////////////////////////////////////////////////////////////////////////
int
DgResourceCell::process (void)
//
// called each iteration
//
{
DgLocation tmpLoc(location());
const DgBoundedRFS2D& brfs = simDB_->physicalRFS2D().boundedRFS2D();
const TcIVec2D coord = (brfs.discRFS().getAddress(tmpLoc))->address();
const DgBoundedRF2D& brf = *(brfs.grids()[0]);

static int timeStep =0;
static int  meanRain = 0; // meanRain is randomly sampled from the vector of 42-year annual
rainfall

if (coord == TcIVec2D(0,0)){
        sprintf(out_string,"%d %d %f %f\n",timeStep, meanRain, lambda, totalPopulation);
        simDB()->outFile() <<"  "<<out_string;

        totalPopLastYear = totalPopulation;
        timeStep++;
        if ((timeStep % 10) == 0)
                cout <<timeStep<<" % done"<<endl;

        if (((timeStep-1) % outCellFreq) == 0) {
            char foutName[30];
                sprintf (foutName,"%s.%d",outCellFileName, timeStep);
                fout.open (foutName, ios::out);
        }

        totalPopulation = 0;
        totalSize0 = 0;
        totalSize1 = 0;
        totalSize2 = 0;
        totalSize3 = 0;
        totalSize4 = 0;
        totalSize5 = 0;
        totalSize6 = 0;
        totalSize7 = 0;

}
//////////// START MOVING

   double minResPressure = varState().nIndividuals() /
                   (varState().nResource()+0.01);
   double curResPressure =  varState().nIndividuals() /
                          (varState().nResource()+0.01);

   DgResourceCell* maxCell = 0;

   for (int i = 0; i < neighbors().size(); i++)
   {
      const DgResourceCell* cell = dynamic_cast<const DgResourceCell*>(neighbors()[i]);

      if (!cell) continue;

      double newResPressure = cell->varState().nIndividuals()/(cell-
>varState().nResource()+0.01);
      if (newResPressure < minResPressure && cell->varState().nResource() >=1)
      {
        minResPressure = newResPressure;
        maxCell = const_cast<DgResourceCell*>(cell);
      }
   }


   if (maxCell != 0)
{
      double diff =  curResPressure - minResPressure;
      double nMoving = 0;

        if (MovingEqua == 1) {
                nMoving = .01 * varState().nIndividuals() * diff;}
        else if (MovingEqua == 2) {
                nMoving = .03 * varState().nIndividuals() * diff;}
        else if (MovingEqua == 3) {
                nMoving = .06 * varState().nIndividuals() * diff;}
        else if (MovingEqua == 4) {
                nMoving = .10 * varState().nIndividuals() * diff;}
        else
                {exit(1);}


double sumWeights = mobilitySize0 * varState().nSize0() +
                                mobilitySize1 * varState().nSize1() +
                                mobilitySize2 * varState().nSize2() +
                                mobilitySize3 * varState().nSize3() +
                                mobilitySize4 * varState().nSize4() +
                                mobilitySize5 * varState().nSize5() +
                                mobilitySize6 * varState().nSize6() +
                                mobilitySize7 * varState().nSize7() ;
        double nMovingSize0 = (mobilitySize0*varState().nSize0() / (sumWeights+1)) * nMoving;
      double nMovingSize1 = (mobilitySize1*varState().nSize1() / (sumWeights+1)) * nMoving;
      double nMovingSize2 = (mobilitySize2*varState().nSize2() / (sumWeights+1)) * nMoving;
      double nMovingSize3 = (mobilitySize3*varState().nSize3() / (sumWeights+1)) * nMoving;
      double nMovingSize4 = (mobilitySize4*varState().nSize4() / (sumWeights+1)) * nMoving;
      double nMovingSize5 = (mobilitySize5*varState().nSize5() / (sumWeights+1)) * nMoving;
      double nMovingSize6 = (mobilitySize6*varState().nSize6() / (sumWeights+1)) * nMoving;
      double nMovingSize7 = (mobilitySize7*varState().nSize7() / (sumWeights+1)) * nMoving;

      const DgResourceCell* cell = dynamic_cast<const DgResourceCell*>(maxCell);

if (nMoving > 0.0)
        {
if (nextVarState().nSize0() > nMovingSize0)
        nextVarState().setNSize0(nextVarState().nSize0() - nMovingSize0);
else
```

```
        {
                nMovingSize0 = nextVarState().nSize0();
                nextVarState().setNSize0(0);
        }

    if (nextVarState().nSize1() > nMovingSize1)
            nextVarState().setNSize1(nextVarState().nSize1() - nMovingSize1);
    else
        {
                nMovingSize1 = nextVarState().nSize1();
                nextVarState().setNSize1(0);
        }
    if (nextVarState().nSize2() > nMovingSize2)
            nextVarState().setNSize2(nextVarState().nSize2() - nMovingSize2);
    else
        {
                nMovingSize2 = nextVarState().nSize2();
                nextVarState().setNSize2(0);
        }

    if (nextVarState().nSize3() > nMovingSize3)
            nextVarState().setNSize3(nextVarState().nSize3() - nMovingSize3);
    else
        {
                nMovingSize3 = nextVarState().nSize3();
                nextVarState().setNSize3(0);
        }

    if (nextVarState().nSize4() > nMovingSize4)
            nextVarState().setNSize4(nextVarState().nSize4() - nMovingSize4);
    else
        {
                nMovingSize4 = nextVarState().nSize4();
                nextVarState().setNSize4(0);
        }

    if (nextVarState().nSize5() > nMovingSize5)
            nextVarState().setNSize5(nextVarState().nSize5() - nMovingSize5);
    else
        {
                nMovingSize5 = nextVarState().nSize5();
                nextVarState().setNSize5(0);
        }

    if (nextVarState().nSize6() > nMovingSize6)
            nextVarState().setNSize6(nextVarState().nSize6() - nMovingSize6);
    else
        {
                nMovingSize6 = nextVarState().nSize6();
                nextVarState().setNSize6(0);
        }
    if (nextVarState().nSize7() > nMovingSize7)
            nextVarState().setNSize7(nextVarState().nSize7() - nMovingSize7);
    else
        {
                nMovingSize7 = nextVarState().nSize7();
                nextVarState().setNSize7(0);
        }
        maxCell->nextVarState().setNSize0(maxCell->nextVarState().nSize0()+nMovingSize0);
        maxCell->nextVarState().setNSize1(maxCell->nextVarState().nSize1()+nMovingSize1);
        maxCell->nextVarState().setNSize2(maxCell->nextVarState().nSize2()+nMovingSize2);
        maxCell->nextVarState().setNSize3(maxCell->nextVarState().nSize3()+nMovingSize3);
        maxCell->nextVarState().setNSize4(maxCell->nextVarState().nSize4()+nMovingSize4);
        maxCell->nextVarState().setNSize5(maxCell->nextVarState().nSize5()+nMovingSize5);
        maxCell->nextVarState().setNSize6(maxCell->nextVarState().nSize6()+nMovingSize6);
        maxCell->nextVarState().setNSize7(maxCell->nextVarState().nSize7()+nMovingSize7);
        maxcell->nextVarstate().setNIndividuals (maxCell->nextVarState().nIndividuals +
          nMovingSize0 + nMovingSize 1+ nMovingSize2 + nMovingSize3 + nMovingSize4+
          nMovingSize5 + nMovingSize6 + nMovingSize7)
          )
}
nextVarState().setNIndividuals (nextVarState().nSize0() + nextVarState().nSize1() +
                                nextVarState().nSize2() + nextVarState().nSize3() +
                                nextVarState().nSize4() + nextVarState().nSize5() +
                                nextVarState().nSize6() + nextVarState().nSize7() );
 /////////////END OF MOVING
```

```
//////////////START OF DEATH & REPRODUCTION

double rain0, rainl, rain2, rain3, rain4, rain5, rain6, rain7, rain8, rain9;

/////Start sampling the rainfall for this year

if (coord == TcIVec2D(0,0)){
        Vector rainYear (42);
        rainYear = rainVec;
        meanRain = randinVector (rainYear);

        rain0 = 0.2 * meanRain;
        rainl = 0.4 * meanRain;
        rain2 = 0.6 * meanRain;
        rain3 = 0.8 * meanRain;
        rain4 = 1.0 * meanRain;
        rain5 = 1.0 * meanRain;
        rain6 = 1.2 * meanRain;
        rain7 = 1.4 * meanRain;
        rain8 = 1.6 * meanRain;
        rain9 = 1.8 * meanRain;




if (rainPattern == 1) {

/*Vector rainCellVector (30);

                           -     -
                     -     -     -     -
               -     -     -     -     -     -
         -     -     -     -     -     -     -     -
   -     -     -     -     -     -     -     -     -     -
 rain0 rainl rain2 rain3 rain4 rain5 rain6 rain7 rain8 rain9

*/

        int i;
                rainCellVector.put_value (0, rain0);

        for (i =1; i<3; i++)
                rainCellVector.put_value (i, rainl);
        for (i =3; i<6; i++)
                rainCellVector.put_value (i, rain2);
        for (i =6; i<10; i++)
                rainCellVector.put_value (i, rain3);
        for (i =10; i<15; i++)
                rainCellVector.put_value (i, rain4);
        for (i =15; i<20; i++)
                rainCellVector.put_value (i, rain5);
        for (i =20; i<24; i++)
                rainCellVector.put_value (i, rain6);
        for (i =24; i<27; i++)
                rainCellVector.put_value (i, rain7);
        for (i =27; i<29; i++)
                rainCellVector.put_value (i, rain8);
                rainCellVector.put_value (29, rain9);

}
if (rainPattern == 2) {

/*Vector rainCellVector (30);

   -     -     -     -     -     -     -     -     -     -
   -     -     -     -     -     -     -     -     -     -
   -     -     -     -     -     -     -     -     -     -
 rain0 rainl rain2 rain3 rain4 rain5 rain6 rain7 rain8 rain9

*/
int i;
        for (i =0; i<3; i++)
                rainCellVector.put_value (i, rain0);
        for (i =3; i<6; i++)
                rainCellVector.put_value (i, rain1);
        for (i =6; i<9; i++)
                rainCellVector.put_value (i, rain2);
        for (i =9; i<12; i++)
                rainCellVector.put_value (i, rain3);
        for (i =12; i<15; i++)
                rainCellVector.put_value (i, rain4);
        for (i =15; i<18; i++)
                rainCellVector.put_value (i, rain5);
        for (i =18; i<21; i++)
                rainCellVector.put_value (i, rain6);
        for (i =21; i<24; i++)
                rainCellVector.put_value (i, rain7);
        for (i =24; i<27; i++)
                rainCellVector.put_value (i, rain8);
        for (i =27; i<30; i++)
                rainCellVector.put_value (i, rain9);

}
} //END of rainfall sampling



matrix b(8,1) ;
b.replace( nextVarState().nSize0(),
           nextVarState().nSize1(),
           nextVarState().nSize2(),
           nextVarState().nSize3(),
           nextVarState().nSize4(),
           nextVarState().nSize5(),
           nextVarState().nSize6(),
           nextVarState().nSize7());

matrix pm (8,8);
pm = Leslie;

double rainCellValue = randinVector (rainCellVector);

nextVarState().setRainCell (rainCellValue);

double rainFactor = 1; //Represent the site-specific effect of rainfall on reproduction and
survival
double roadFactor = 1; //Represent the site-specific effect of roads on reproduction and
survival
double combinedFactor = 1; //Combined effect of those two factors.

//MEAN_ROAD and MEAN_RAIN are defined in DgResourceSim.h

if      (RepSurEquaRoad == 1) {
        roadFactor = 1.0 - (0.01 * (varState().nRoad() - MEAN_ROAD) / MEAN_ROAD);
}

else if (RepSurEquaRoad == 2) {
        roadFactor = 1.0 - 0.02 * (varState().nRoad() - MEAN_ROAD) / MEAN_ROAD;
}

else if (RepSurEquaRoad == 3) {
        roadFactor = 1.0 - 0.03 * (varState().nRoad() - MEAN_ROAD) / MEAN_ROAD;
}

else if (RepSurEquaRoad == 4) {
        roadFactor = 1.0 - 0.04 * (varState().nRoad() - MEAN_ROAD) / MEAN_ROAD;
}

else {roadFactor = 1.0;}

if (RepSurEquaRain == 1) {
        rainFactor = 1.0 + (0.01 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
}

else if (RepSurEquaRain == 2) {
        rainFactor = 1.0 + (0.02 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
```

```
}

else if (RepSurEquaRain == 3) {
        rainFactor = 1.0 + (0.03 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
}

else if (RepSurEquaRain == 4) {
        rainFactor = 1.0 + (0.04 * (rainCellValue - MEAN_RAIN) / MEAN_RAIN);
}

else {rainFactor = 1.0;}

if (roadFactor < 0)
        roadFactor = 0;

if (rainFactor < 0)
        rainFactor = 0;

combinedFactor = rainFactor * roadFactor;


double resourcePressure = nextVarState().nIndividuals() / (varState().nResource()+0.001);

if (resourcePressure > 6)
        combinedFactor = 1.0 / 0.982; //The population will not grow any more. Lambda of the
cell is 1.

///////////////


if (combinedFactor > 1.07)
        combinedFactor = 1.07; //for reasonable variation of the projection matrix.

pm = pm * combinedFactor;

b = pm * b;

double s0,s1,s2,s3,s4,s5,s6,s7;
s0 = b.get_value (0,0);
s1 = b.get_value (1,0);
s2 = b.get_value (2,0);
s3 = b.get_value (3,0);
s4 = b.get_value (4,0);
s5 = b.get_value (5,0);
s6 = b.get_value (6,0);
s7 = b.get_value (7,0);


//////////////Update size classes after death & reproduction

        nextVarState().setNSize0 (s0);
        nextVarState().setNSize1 (s1);
        nextVarState().setNSize2 (s2);
        nextVarState().setNSize3 (s3);
        nextVarState().setNSize4 (s4);
        nextVarState().setNSize5 (s5);
        nextVarState().setNSize6 (s6);
        nextVarState().setNSize7 (s7);
      nextVarState().setNIndividuals(s0+s1+s2+s3+s4+s5+s6+s7);

//////////// END OF DEATH & REPRODUCTION

////////////START WRITING CELL-SPECIFIC OUTPUTS TO FILES

if (((timeStep-1) % outCellFreq) == 0) {
sprintf(outCell_string,"%d %f %f %f\n",varState().hexId(),
varState().nRoad(),nextVarState().rainCell(),nextVarState().nIndividuals());
        fout<<outCell_string;
        if (coord == TcIVec2D (brf.upperRight().i(), brf.upperRight().j()))
                fout.close();
}
///////////END OF WRITING CELL-SPECIFIC OUTPUTS TO FILES

////////////START CALCULATING POPULATION STATISTICS

totalPopulation = totalpopulation + varState().nIndividuals();
```

```
if (coord == TcIVec2D (brf.upperRight().i(), brf.upperRight().j())) (
        lambda = totalPopulation/(totalPopLastYear+1);

)
///////////END OF CALCULATING POPULATION STATISTICS

    return 0;
) // DgResourceCell::process

//////////////////////////////////////////////////////////////////////////////
int
DgResourceCell::setGraphicState (void)
//
// called before cells are drawn
//
{
   static vector<TcColor*> cols;
   static bool first = true;
   if (first)
   (
      TcColor::linearSpread(TcColor("#ffffff"), TcColor("#0000ff"), cols, 50);
      first = false;)
   setDrawPoint(false);
   setDrawBoundary(true);

   int nVal = (int) ((varState().nIndividuals() / 1500.0)  * 50);
   if (nVal >= 50) nVal = 49;

   const TcColor& fillCol = *cols[nVal];
   boundary().setFillColor(fillCol);

   return 0;

) // DgResourceCell::setGraphicState

//////////////////////////////////////////////////////////////////////////////
int
DgResourceCell::setOutputState (void)
//
//called before cells are output
//
{
   return 0;

} // DgResourceCell::setOutputState



//////////////////////////////////////////////////////////////////////////////
//
// End of DgResourceCell.C:
//
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
//
// DgResourceSim.C:
// This file is for building simulator based on structures and methods of simulation cell.
//
//////////////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;

#include "DgResourceCell.h"
#include "DgResourceSim.h"
#include "TcColor.h"
#include "DgBoundedRFS2D.h"

//////////////////////////////////////////////////////////////////////////////
DgResourceSim::DgResourceSim (DgPhysicalRFS2D<DgResourceCell>& rfIn,
                  double nIndiv,  double nSize0Indiv,
                  double nSize1Indiv, double nSize2Indiv, double nSize3Indiv,
                  double nSize4Indiv, double nSize5Indiv, double nSize6Indiv,
                  double nSize7Indiv, bool useGraphicsIn, bool useOutputIn,
                 const string& outFileNameIn)
     : DgSimDB2DS<DgResourceCell>(rfIn, useGraphicsIn, useOutputIn, outFileNameIn),
     nIndiv_ (nIndiv),
     nSize0Indiv_(nSize0Indiv), nSize1Indiv_(nSize1Indiv),
     nSize2Indiv_(nSize2Indiv), nSize3Indiv_(nSize3Indiv),
     nSize4Indiv_(nSize4Indiv), nSize5Indiv_(nSize5Indiv),
     nSize6Indiv_(nSize6Indiv), nSize7Indiv_(nSize7Indiv)
{

   TcColor::linearSpread(TcColor("#ffffff"), TcColor("#ff0000"), cols_, 50);

} // DgResourceSim::DgResourceSim

////////////////////////////////////////////////////////////////////////////////
int
DgResourceSim::initialize (void)
{
   if (DgSimDB2DS<DgResourceCell>::initialize()) return -1;

   return reset();

} // DgResourceSim<S, V>::initialize

////////////////////////////////////////////////////////////////////////////////
int
DgResourceSim::reset (void)
{
   if (DgSimDB2DS<DgResourceCell>::reset()) return -1;
   srand (randSeed);

   //Read the input files of resource, road and hex-ID to Vectors for further retrieval:

   Vector resourceVec (nHex);
   readfile_to_vec (resourceVec, resFileName);

   Vector roadVec (nHex);
   readfile_to_vec (roadVec, roadFileName);

   Vector hexIdVec (nHex);
   readfile_to_vec (hexIdVec, hexIdFileName);

   //////////Read rainfall data:

   readfile_to_vec (rainVec, "rain.txt");

   //////////Specify mobility for seven size classes.

   if (mobilityPattern == 2) {

        mobilitySize0 = 1;
        mobilitySize1 = 3;
        mobilitySize2 = 5;
        mobilitySize3 = 5;
        mobilitySize4 = 5;
        mobilitySize5 = 2;
        mobilitySize6 = 2;
        mobilitySize7 = 1;
    }
/////////Read Leslie matrix:
Leslie.projection_matrix ();
cout << "Projection matrix :\n";
Leslie.print();

//////////Read information contained in Vectors of resource, road,and hex-ID to hexagons
   DgSpatialDB<DgResourceCell>::iterator it(*this);

   hexNumber = 0;
   for (it = begin(); it != end(); ++it)
   {

      DgLocation tmpLoc((*it)->location());
      const DgBoundedRFS2D& brfs = physicalRFS2D().boundedRFS2D();
        const TcIVec2D coord = (brfs.discRFS().getAddress(tmpLoc))->address();
        const DgBoundedRF2D& brf = *(brfs.grids()[0]);

      (*it)->varState().setNRoad(roadVec.get_value(hexNumber)) ;
```

```
        (*it)->varState().setHexId (hexIdVec.get_value(hexNumber));

if (gridResolution == 14) (
        (*it)->varState().setNResource((resourceVec.get_value(hexNumber)
                               *(1 - (*it)->varState().nRoad() / 100))*10.68832) ;
}

if (gridResolution == 16) (
        (*it)->varState().setNResource((resourceVec.get_value(hexNumber)
                        *(1 - (*it)->varState().nRoad() / 100))*1.18759) ;
}


if ((*it)->varState().nResource() >= 1)
        (*it)->varState().setNIndividuals((*it)->varState().nResource()*4);
else
        (*it)->varState().setNIndividuals(0);

        (*it)->varState().setNSize0((472.0/1907.0) * (*it)->varState().nIndividuals());
        (*it)->varState().setNSize1((815.0/1907.0) * (*it)->varState().nIndividuals());
        (*it)->varState().setNSize2((292.0/1907.0) * (*it)->varState().nIndividuals());
        (*it)->varState().setNSize3((115.0/1907.0) * (*it)->varState().nIndividuals());
        (*it)->varState().setNSize4((64.0/1907.0) * (*it)->varState().nIndividuals());
        (*it)->varState().setNSize5((48.0/1907.0) * (*it)->varState().nIndividuals());
        (*it)->varState().setNSize6((90.0/1907.0) * (*it)->varState().nIndividuals());
        (*it)->varState().setNSize7((11.0/1907.0) * (*it)->varState().nIndividuals());

        totalPopulation = totalPopulation + (*it)->varState().nIndividuals();
        (*it)->nextVarState() = (*it)->varState();
/////////////////////////
        hexNumber = hexNumber + 1;
   }
return 0;

} // DgResourceSim::reset

////////////////////////////////////////////////////////////////////////////////
int
DgResourceSim::setOutputState (void)
{
   return DgSimDB2DS::setOutputState();

} //DgResourceSim::setOutputState

////////////////////////////////////////////////////////////////////////////////
void
DgResourceSim::output (void)
{ } //DgResourceSim::output

////////////////////////////////////////////////////////////////////////////////////////
//
// End of DgResourceSim.C:
//
////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////
//
// oglresource.C and elasticity.C
//
//These two files are main program, in which the files DgResourceCell.h, //DgResourceCell.C,
//DeReourceSim.h, DgResourceSim.C are used. These files are also written for //parameterizing
//the model and used for creating executable files to run model's simulations.
//Note that the code below is of elasticity.C, which has 21 parameters (labeled from 1 to //21)
//The code for oglresource.C is the same as elasticity.C except it has only 19 parameters
//(from 1 to 19).
//
//////////////////////////////////////////////////////////////////////////////////////////


#include <iostream>

using namespace std;
#include <stdlib.h>

#include "DgContCartRF.h"
#include "DgSqrD8Grid2DS.h"
```

```
#include "DgDmdD8Grid2DS.h"
#include "DgHexGrid2DS.h"
#include "TcMIFDisplay.h"
#include "DgResourceSim.h"
#include "DgBoundedRFS2D.h"
#include "DgPhysicalRFS2D.h"

////////////////////////////////////////////////////////////////////////////////
int main (int argc, char* argv[])
{
   // process command line arguments

   TcBase::testArgEqual(argc, argv, 21,
          string("1.hex|tri|sqr4|sqr8  2.maxX  3.maxY  4.saveFreq  5.stopTime \n"
                 "6.resourceFile  7.roadFile  8.hexIdFile  9.rainPattern  10.mobilityPatern\n"
                 "11.MovingEqua#  12.RepSurEquaRoad  13.RepSurEquaRain  14.RandSeed\n"
"15.gridResolution  "
                 "16.useGraphics 17.useOutput [18.outFileName] 19.outputCellFreq \n 20.pm#
21.percentChange"));
   string geom(argv[1]);
   TcIVec2D lowerLeft(0, 0);
   TcIVec2D upperRight(atoi(argv[2]), atoi(argv[3]));
   nHex = (atoi(argv[2]) + 1) * (atoi(argv[3]) + 1);
   int saveFreq(atoi(argv[4])); // 0 indicates never save
   int stopTime(atoi(argv[5]));
   resFileName = argv[6];
   roadFileName = argv[7];
   hexIdFileName = argv[8];
   rainPattern = atoi(argv[9]);
   mobilityPattern = atoi (argv[10]);
   MovingEqua = atoi (argv[11]);
   RepSurEquaRoad = atoi (argv[12]);
   RepSurEquaRain = atoi (argv[13]);
   randSeed = atoi (argv[14]);
   gridResolution = atoi (argv[15]);

   bool useGraphics = (atoi(argv[16]) == 1);
   bool useOutput = (atoi(argv[17]) == 1);

   string outFileName;
   if (argc > 10) outFileName = argv[18];
   outCellFileName = argv[18];
   outCellFreq = atoi(argv[19]);
   int pm_index = atoi(argv[20]);
   double percentChange = atof(argv[21]);

if      (pm_index == 1)
        pm05 = pm05 * (1 + (percentChange/100.0));
else if (pm_index == 2)
        pm06 = pm06 * (1 + (percentChange/100.0));
else if (pm_index == 3)
        pm07 = pm07 * (1 + (percentChange/100.0));
else if (pm_index == 4)
        pm10 = pm10 * (1 + (percentChange/100.0));
else if (pm_index == 5)
        pm11 = pm11 * (1 + (percentChange/100.0));
else if (pm_index == 6)
        pm21 = pm21 * (1 + (percentChange/100.0));
else if (pm_index == 7)
        pm22 = pm22 * (1 + (percentChange/100.0));
else if (pm_index == 8)
        pm32 = pm32 * (1 + (percentChange/100.0));
else if (pm_index == 9)
        pm33 = pm33 * (1 + (percentChange/100.0));
else if (pm_index == 10)
        pm43 = pm43 * (1 + (percentChange/100.0));
else if (pm_index == 11)
        pm44 = pm44 * (1 + (percentChange/100.0));
else if (pm_index == 12)
        pm54 = pm54 * (1 + (percentChange/100.0));
else if (pm_index == 13)
        pm55 = pm55 * (1 + (percentChange/100.0));
else if (pm_index == 14)
        pm65 = pm65 * (1 + (percentChange/100.0));
else if (pm_index == 15)
        pm66 = pm66 * (1 + (percentChange/100.0));
```

```
else if (pm_index == 16)
        pm76 = pm76 * (1 + (percentChange/100.0));
else if (pm_index == 17)
        pm77 = pm77 * (1 + (percentChange/100.0));
else { }

//////////// build the simulator

   DgRFNetwork net;
   DgContCartRF cc0(net, "ContCart0");

   DgDiscRFS2D* dg0;
   if (geom == "hex")
   {
      dg0 = new DgHexGrid2DS(net, cc0, 1, 4, false, true, "HexC12DS");
   }

   else
   {
      report(string(argv[0]) + "() bad or unimplemented geometry type: "
             + geom, TcBase::Fatal);
   }

if (rainPattern != 1 && rainPattern != 2) {
        cout <<"rainPattern# is not included.\n";
        cout <<"Let choose 1 or 2.\n";
        exit(1);
}

if (mobilityPattern != 1 && mobilityPattern != 2) {
        cout <<"mobilityPattern# is not included.\n";
        cout <<"Let choose 1 or 2.\n";
        exit(1);
}

if (MovingEqua != 1 && MovingEqua !=2 &&
    MovingEqua != 3 && MovingEqua !=4) {
        cout<<"Moving Equation # is not included.\n";
        cout <<"Let choose 1, 2, 3 or 4.\n";
        exit(1);
}

if (RepSurEquaRoad != 0 && RepSurEquaRoad != 1 && RepSurEquaRoad != 2 &&
RepSurEquaRoad != 3 && RepSurEquaRoad != 4 ) {
        cout <<"RepSurEquaRoad is not included.\n";
        cout <<"Let choose 0, 1, 2, 3 or 4.\n";
        exit(1);
}

if (RepSurEquaRain != 0 && RepSurEquaRain != 1 && RepSurEquaRain != 2 &&
    RepSurEquaRain != 3 && RepSurEquaRain != 4 ) {
        cout <<"RepSurEquaRain is not included.\n";
        cout <<"Let choose 0, 1, 2, 3 or 4.\n";
        exit(1);
}
if (gridResolution != 14 && gridResolution != 16) {
        cout <<"The specified grid resolution is not included.\n";
        cout <<"Let choose 14 or 16.\n";
        exit(1);
}

   DgBoundedRFS2D b0(*dg0, lowerLeft, upperRight);
   DgPhysicalRFS2D<DgResourceCell> p0(b0);

   TcMIFDisplay display("MIF");

   DgResourceSim resource1(p0,0,0,0,0,0,0,0,0,0,useGraphics, useOutput,outFileName);
   resource1.setStopTime(stopTime);
   resource1.setSaveFreq(saveFreq);
   StopTime = stopTime;

cout <<"\n Running simulation with: \n"
        <<" nHex: "<<nHex<<"\n upperRight: "<<upperRight<<"\n lowerLeft: "<<lowerLeft<<"\n"
        <<" resFileName: "<<resFileName<<"\n roadFileName: "<<roadFileName<<"\n hexIdFileName:
"
        << hexIdFileName<<"\n"
        <<" rainPattern = "<<rainPattern<<"\n mobilityPattern = "<<mobilityPattern<<"\n"
        <<" MovingEqua = "<<MovingEqua<<"\n RepSurEquaRoad = "<<RepSurEquaRoad
        <<" \n RepSurEquaRain = "<<RepSurEquaRain<<"\n"
        <<" gridResolution = "<<gridResolution<<"\n outCellFileName: "<<outCellFileName
        <<" \n outCellFreq = "<<outCellFreq<<"\n";

   resourcel.initialize();
   resourcel.simulate();
   resourcel.postProcess();

   return 0;

}
////////////////////////////////////////////////////////////////////////////////////////////
//
//End of oglresource.C and elasticity.C
//
////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////
//
//rmm.C
//This file is to calculate the eigenvalues, eigenvectors, and elasticities
//of the projection matrix.
//Header files are from a C++ template matrix library developed by ***
//
////////////////////////////////////////////////////////////////////////////////////////////

#include <iostream>

#include <complex>

#include "mtl/matrix.h"
#include "mtl/mtl.h"
#include "mtl/utils.h"
#include "mtl/matrix_market_stream.h"
#include "mtl/mtl2lapack.h"
#include "mtl/denselD.h"
#include "mtl/mmio.c"

int
main()
{
  using namespace mtl;
  using namespace mtl2lapack;

  matrix_market_stream <double>  mms("test.mm");
  lapack_matrix<double>::type A(mms);

  int N = A.ncols();

  lapack_matrix<double>::type AA(N,N);
  copy (A,AA);
  lapack_matrix<double>::type vr(N,N);
  lapack_matrix<double>::type vl(N,N);
  mtl::denselD< std::complex<double> > wr(N);
  int info;

  // Compute the eigenvalues and eigenvectors of A.

  info = geev(GEEV_CALC_BOTH, A, wr, vl, vr);

  if (info > 0) {
    cout << "QR failed to converge, INFO = " << info << endl;
    return 0;
  }

  // Print the eigenvalues and eigenvectors.

  cout << "eigenvalues" << endl;
  mtl::print_vector(wr);

  cout << "left eigenvectors" << endl;
  mtl::print_all_matrix(vl);

  cout << "right eigenvectors" << endl;
  mtl::print_all_matrix(vr);

  //Extract the dominant left and right eigenvectors to calculate elasticities.

  int imax = max_index(wr);
  cout <<"max index of wr: imax = "<<imax<<endl;
  complex<double> Max_Lambda;
  Max_Lambda = wr[imax];
  double real_lambda = Max_Lambda.real();
  cout <<"real_lambda = "<<real_lambda<<endl;

  typedef matrix<double,rectangle<>,dense<>,row_major>::type mymatrix;
  mymatrix mright(N,N);
  copy (vr, mright); //vr contains all right eigenvectors.

  mymatrix mleft (N,N);
  copy (vl, mleft); //vl contains all left eigenvectors.


mymatrix::submatrix_type sub_right;
mymatrix::submatrix_type sub_left;

sub_right = mright.sub_matrix (0,N,imax,imax + 1); //sub_right is right dominant eigenvector
sub_left = mleft.sub_matrix (0,N,imax,imax + 1); // sub_left is left dominant eigenvector

cout<<"right eigenvector ~ Max_Lambda: \n";
mtl::print_all_matrix (sub_right);

cout<<"left eigenvector ~ Max_Lambda: \n";
mtl::print_all_matrix (sub_left);


////////////////////Calculate elasticities

double sum =0;

for (int i =0; i<N; i++) {
        sum = sum + sub_right(i,0) * sub_left(i,0);
}
cout <<"\n sum = <w,v> = "<<sum<<endl;

mymatrix Sensitivity(N,N);
for (int i =0; i<N; i++) {
        for (int j =0; j<N;j++)
                Sensitivity(i,j) = sub_left(i,0) * sub_right(j,0) / sum;
}

mymatrix Elasticity (N,N);
for (int i =0; i<N; i++) {
        for (int j =0; j<N;j++)
                Elasticity (i,j) = Sensitivity (i,j) * AA(i,j) / real_lambda;
}



cout <<"Sensitivity matrix: \n";
mtl::print_all_matrix (Sensitivity);

cout <<"Elasticity matrix: \n";
mtl::print_all_matrix(Elasticity);

/////////////////////////////////////////////////////////////////////////////////////////
//
//End of rmm.C
//
/////////////////////////////////////////////////////////////////////////////////////////
```