INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road. Ann Arbor. MI 48106-1346 USA
313/761-4700 800/521-0600

Habitat Connectivity And Spotted Owl Population Dynamics

by

Nathan Halsey Schumaker

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

1995

Approved by ______________________________
Co-Chairperson of Supervisory Committee

____
Co-Chairperson of Supervisory Committee

Jerry F. Franklin.
____

____

Program Authorized
to Offer Degree College of Forest Resources

Date February 1, 1995

UMI Number: 9537355

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction for this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform"

Signature ______________________

Date February 1, 1995

University Of Washington

Abstract

Habitat Connectivity and Spotted Owl Population Dynamics

by Nathan Halsey Schumaker

Chairpersons of The Supervisory Committee: Professor Stephen D. West
College of Forest Resources
Professor Peter M. Kareiva
Department of Zoology

The ecological consequences of habitat fragmentation include the direct effects of habitat loss and the indirect effects of reduced inter-patch dispersal. I examine these consequences of habitat fragmentation through two separate studies. The first study measures habitat connectivity through the success of a simulated dispersal process, and then asks if connectivity can be estimated from measures of habitat pattern alone. The second study examines the effect of past habitat loss, and of potential future habitat gains, on an isolated population of northern spotted owls on the Olympic Peninsula of Washington state.

Indices of landscape pattern are frequently used to estimate habitat connectivity, but whether they actually do so remains undocumented. If indices of habitat pattern do indeed estimate habitat connectivity, then these indices should correlate well with predictions of dispersal success. To test this possibility, I looked for correlations between nine common indices of habitat pattern and the results of a simulated dispersal process conducted using GIS data for old-growth forest throughout the Pacific Northwest. The nine indices of habitat pattern that I examined were only weakly correlated with the results from the dispersal modeling, but I identified a new pattern index, termed patch cohesion, for which the fit was much better.

I constructed a spatially explicit simulation model for the population of spotted owls on the Olympic Peninsula of Washington state, and used this model to examine the future of this population, and how patterns of occupancy might differ in habitats of varying quality. Results from model simulations that incorporate habitat loss over the last three quarters of a century lend support to suspicions that the population of spotted owls on the Olympic Peninsula may presently be in a sharp decline. But simulations that account for regeneration of owl habitat over the next 100 years show that the potential exists for this trend to reverse, and for the population to stabilize. The simulated owl population showed a two-fold response to the addition of new habitat over what would have been expected based on estimates of habitat area alone.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Number *Page*

LIST OF TABLES

Acknowledgments

This work was supported by USDA/USFS Grant PNW 90-340 to R. J. Naiman, U.S. State Department Grant 1753-000574 to R. J. Naiman, M. G. Turner, and R. G. Lee, NSF Grant BIR9256532 to G. Odell, T. Daniel, and P. Kareiva, and by Washington Department of Natural Resources Interagency Agreement FY94-053 to Steve West. I developed the initial ideas for the computer model while attending a summer school on Patch Dynamics organized by S. A. Levin, T. M. Powell, and J. H. Steele and held at Cornell University in 1991. I thank Steve West, Mark Kot, Dan Doak, Steve Minta, Danny Grünbaum, Tom Daniel, Garry Odell, and above all Peter Kareiva for painstaking rectification of both ideas and words. Much of this work was conducted at the Center for Conservation Biology at University of California at Santa Cruz, where I made extensive use of the GIS facility managed by the Board of Environmental Studies. In addition, the study presented in chapter 1 benefited from lengthy discussions with Steve Minta and Dan Doak of the UCSC Board of Environmental Studies. The analysis described in chapter 2 was improved through the involvement of Scott Horton and George Wilhere of the Washington Department of Natural Resources. Peter Kareiva and Steve West provided the endless encouragement without which none of this work would have been possible. Lastly, I'd like to thank my family, Dawn Arthur, Dave Leversee, and Dave & Suzan Shaw for providing refuge from the hazards of academia.

Introduction

The ecological consequences of habitat fragmentation include the direct effects of habitat loss and the indirect effects of reduced inter-patch dispersal. Elimination of habitat reduces the carrying capacity of an affected landscape. At the same time, habitat patches that survive the process of fragmentation become increasingly isolated from one-another, and this can cause species' declines in excess of what would be predicted based strictly on reductions in habitat area. In this dissertation, I examine both of these consequences of habitat fragmentation through two separate studies. The first study measures habitat connectivity through the success of a simulated dispersal process, and then asks if the results could be predicted based on habitat pattern alone. The second study examines the effect of past habitat loss, and of potential future habitat gains, on an isolated population of northern spotted owls on the Olympic Peninsula of Washington state. I briefly outline these studies below.

To quantify the hinderance of dispersal caused by habitat fragmentation, landscape ecologists have invented the notion of habitat connectivity. Indices of landscape pattern are frequently used to estimate habitat connectivity, but whether they actually do so remains undocumented. If indices of habitat pattern do indeed estimate habitat connectivity, then these indices should correlate well with predictions of dispersal success. To test this possibility, I looked for correlations between nine common indices of habitat pattern and the results of a simulated dispersal process conducted using geographic information system (GIS) data for old-growth forest throughout the Pacific Northwest. The nine indices of habitat pattern that I examined were only weakly correlated with the results from the dispersal modeling, but I identified a new pattern index, termed patch cohesion, for which the fit was much better. Chapter 1 of this dissertation presents a complete description of this connectivity study.

The northern spotted owl is an ideal subject for studies of the effects of habitat fragmentation because it responds strongly to losses of both habitat area and connectivity. To examine the interactions between these two consequences of habitat fragmentation, I constructed a spatially explicit simulation model for the population of spotted owls on the Olympic Peninsula of Washington state. I used this model to examine the future of this population and how patterns of occupancy might differ in habitats of varying quality. Results from model simulations that incorporate habitat loss over the last three quarters of a century lend support to suspicions that the population of spotted owls on the Olympic Peninsula may presently be in a sharp decline. But simulations that account for regeneration of owl habitat over the next 100 years show the potential exists for this trend to reverse, and for the population to stabilize. Further, the simulated owl population showed a response to the addition of new habitat that exceeded twice what would have been expected based on estimates of carrying capacity. This response to the recruitment of new habitat demonstrated the strong relationship that exists between spotted owl population trends and trends in habitat connectivity. My research into northern spotted owl population dynamics appears in chapter 2 of this dissertation.

Both of the studies described in this dissertation were conducted using a computer model that I constructed explicitly for this use. The computer model reads raster GIS habitat data and contains separate modules that quantify habitat pattern, conduct dispersal trials, and track the fate of a simulated population of spotted owls. The model was designed to carefully examine relationships between habitat pattern and ecological processes. This goal was met by keeping the model simple biologically, while paying a great deal of attention to interactions between the simulated organisms and the landscapes they inhabited. Great care was taken to ensure that the model works correctly. I have reproduced portions of the computer code for the movement and spotted owl demography modules in appendices 1 and 2, at the end of this dissertation. This text makes up 10% of the model code (excepting code that exists only to create the graphical user interface) and is intended to benefit readers involved in the construction of similar models, and those interested in the mechanisms that accomplish tasks described in chapters 1 and 2. Interested readers may obtain a free copy of the complete source code from the author.

Chapter 1: Using Landscape Indices To Predict Habitat Connectivity

Introduction

The decline of many species has been linked directly to habitat loss and fragmentation. Conservation strategies now frequently consider not only amounts of habitat that must be retained, but also the spatial configurations of habitat across landscapes of concern (e.g., Thomas et al. 1990, Pulliam et al. 1992). The growing accessibility of remotely sensed data and geographical information system (GIS) software has encouraged this trend, and has focused attention on the application of landscape ecology theory to conservation biology (Doak and Mills 1994, Groom and Schumaker, 1993). Much of the effort to integrate conservation biology with landscape ecology has concentrated on the development of better methods for quantifying habitat fragmentation (e.g., Turner 1989, Franklin and Forman 1987, Ripple et al. 1991, O'Neill et al. 1988), and on the construction of realistic spatially-explicit demographic models (McKelvey 1993, Pulliam et al. 1992). But a paucity of life-history data dictates that, for all but the best understood species, spatially explicit demographic models will be of limited utility (Doak and Mills 1994). In contrast, any number of indices of habitat fragmentation can be calculated easily and reliably using today's technology and data. Numerous indices of landscape pattern have been linked to ecological function (Franklin and Forman 1987, Hansen and Urban 1992, Johnson et al. 1992, Turner et. al. 1989*a*, Ripple et al 1991, Noss 1990, Miller et al. 1989, Shaw and Atkinson 1990, Duever and Noss 1990, Keller 1990, Burel 1989). But the value of such indices has yet to be convincingly illustrated (Groom and Schumaker, 1993) because much of the existing work has involved the use of artificial, computer generated landscapes to examine pattern indices that are intuitively appealing but have no documented relationship to critical ecological phenomena (e.g. O'Neil et al. 1988, Turner et al. 1989*b*, Turner, 1989, Gardner and O'Neill 1991, Gustafson and Parker 1992, Plotnick et al. 1993).

I report on a quantitative examination of nine commonly cited indices of habitat pattern and their ability to predict habitat connectivity. A useful index of habitat connectivity should correlate strongly with the success of a realistic foraging, mate-finding, or dispersal process (Buechner 1989, Stamps et al. 1987, Buechner 1987, Kareiva 1987, Adler and Nuernberger 1994, Taylor et al. 1993, Fahrig and Merriam 1985, Fahrig and Paloheimo 1988*a,b*, Harrison 1989, Hastings and Wolin 1989), but indices encountered in the landscape ecology literature rarely undergo such scrutiny. To rectify this gap in our knowledge, I make use here of a GIS map showing the actual patterns of old-growth forests across the Pacific Northwest, and measure the degree of correlation between pattern indices and the results of a spatially-explicit dispersal model run in these landscapes. I used dispersal success rate to facilitate the comparison of pattern indices because low dispersal success is often tied to population decline in fragmented habitats (Doak et al. 1992, Thomas et al. 1990, Pulliam et al. 1992, Whitcomb et al. 1981, Fahrig and Merriam 1985, Henein and Merriam 1990). By using dispersal success as a gauge for habitat connectivity, I am able to evaluate different descriptive indices of landscape structure. The dispersal model I employed is general but, to the extent data permit, could be parameterized for a specific organism.

Methods

Overview of approach

I conducted this study using computer software that I wrote in C to run on a SUN Workstation™. This model was designed for use in an ongoing investigation into the consequences of habitat fragmentation for the northern spotted owl (*Strix occidentalis caurina*), and some of the analysis described below reflect this original purpose. Several distinct steps were involved in generating the results presented here. I initially selected a large number of sample landscapes from the GIS data set mentioned in the introduction. I then examined every habitat patch in each sample landscape and computed landscape pattern indices using data from all of the patches in the landscape. For my dispersal model, I assumed a territorial organism in search of vacant territories -- this meant I transformed each sample landscape into a map of suitable sites or territories. Lastly, the dispersal modeling itself consisted of simulating a large number of individual movements in search of vacant territories.

Habitat data

The data used in this study provide an estimate of the amount and distribution of old-growth forest in the National Forests of the Pacific Northwest from the Cascade Mountains west to the Pacific Coast, and from the top of Washington state south through northern California (figure 1). The GIS map that I used was developed by The Wilderness Society from Landsat multispectral scanner (MSS) data, aerial photography, and field data. I obtained the data in raster format at a resolution of one pixel = 57×57 meters^2. Only old-growth forests within the National Forest system were included in this data. The Wilderness Society used the PNW-444 definition of "old-growth" (Old-Growth Definition Task Group 1986) in their classification, but not all of the National Forests were mapped with equal effort. This data set is quite large, containing 17,536 rows and 4766 columns, and representing 1.53 million hectares of old-growth forest. The true amount of old-growth forest in the Pacific Northwest is disputed, and depends on the definition used (Marcot et al. 1991). Note that because ownerships other than National Forests lands were not mapped, the data provide a lower bound on the distribution of old-growth forests. However, the majority of the remaining old-growth forest in the Pacific Northwest is found within the National Forest system (at least 83% based on Marcot et al. 1991).

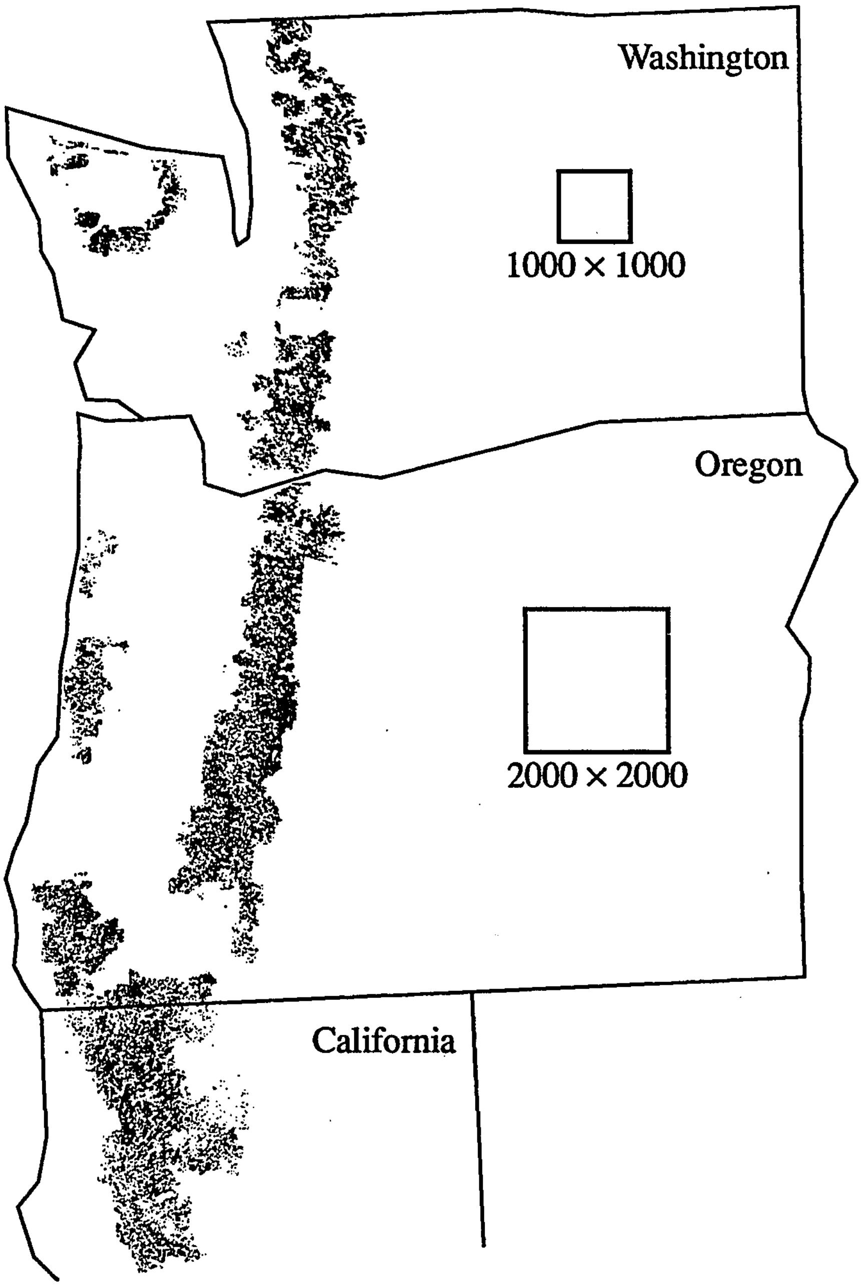


Figure 1. The Wilderness Society GIS data for old-growth forests in the National Forests of the Pacific Northwest, from the Cascade Mountains west to the Pacific Coast. Old-growth forest is black in the figure, and the size of a single data pixel is 57×57 meters. The small and large squares show the size of the 1000×1000 and 2000×2000 pixel sample landscapes used in the study. The state boundaries are only roughly approximated.

Landscape selection

The principal unit of analysis in this study is the landscape, and in lieu of using artificial landscapes, I chose to work with actual habitat patterns to capture some of the spatial heterogeneity present in the natural world. The use of real habitat data also facilitates application of the results and makes the study more useful to ecologists investigating spatially explicit population models, which have recently become quite popular (Kareiva and Wennergren in review, Turner et al. in press, Dunning et al. in press).

The sheer size of The Wilderness Society's old-growth data set allowed me to both employ actual habitat data and retain a large number of replicate landscapes. For the purpose of this study, 1000×1000 and 2000×2000 pixel (3249 and 12,996 km^2, respectively) sample landscapes were chosen at random from the original GIS data set. Figure 1 indicates the relative sizes of these landscapes. A simple algorithm selected row and column boundaries such that the sample landscapes fit entirely within the full data set. A total of 2100 randomly selected landscapes were used to make the comparisons between habitat pattern and dispersal success. A lower limit on the amount of old-growth present in any sample landscape was specified and if this threshold amount of habitat was not present, then the landscape was rejected and another random selection was made. I found this to be a simple and reliable mechanism for eliminating landscapes in which habitat was so sparse as to obscure any relationships between pattern and process. My initial investigations were carried out with a minimum habitat threshold of 1%, but in the tests conducted exclusively of patch connectivity I raised this value to 5%. The 1000×1000 and 2000×2000 pixel landscapes contained up to 33.4% and 17.2% old-growth in up to 12 and 30 thousand separate old-growth patches, respectively. Figure 2 displays four of the 1000×1000 pixel landscapes I used in this study. These landscapes represent between 15% and 22% old-growth forest. Large landscapes, such as these, can be so complex as to preclude meaningful visual assessment of non-trivial attributes like degree of habitat connectivity. For example, though quantification of landscape pattern may not be

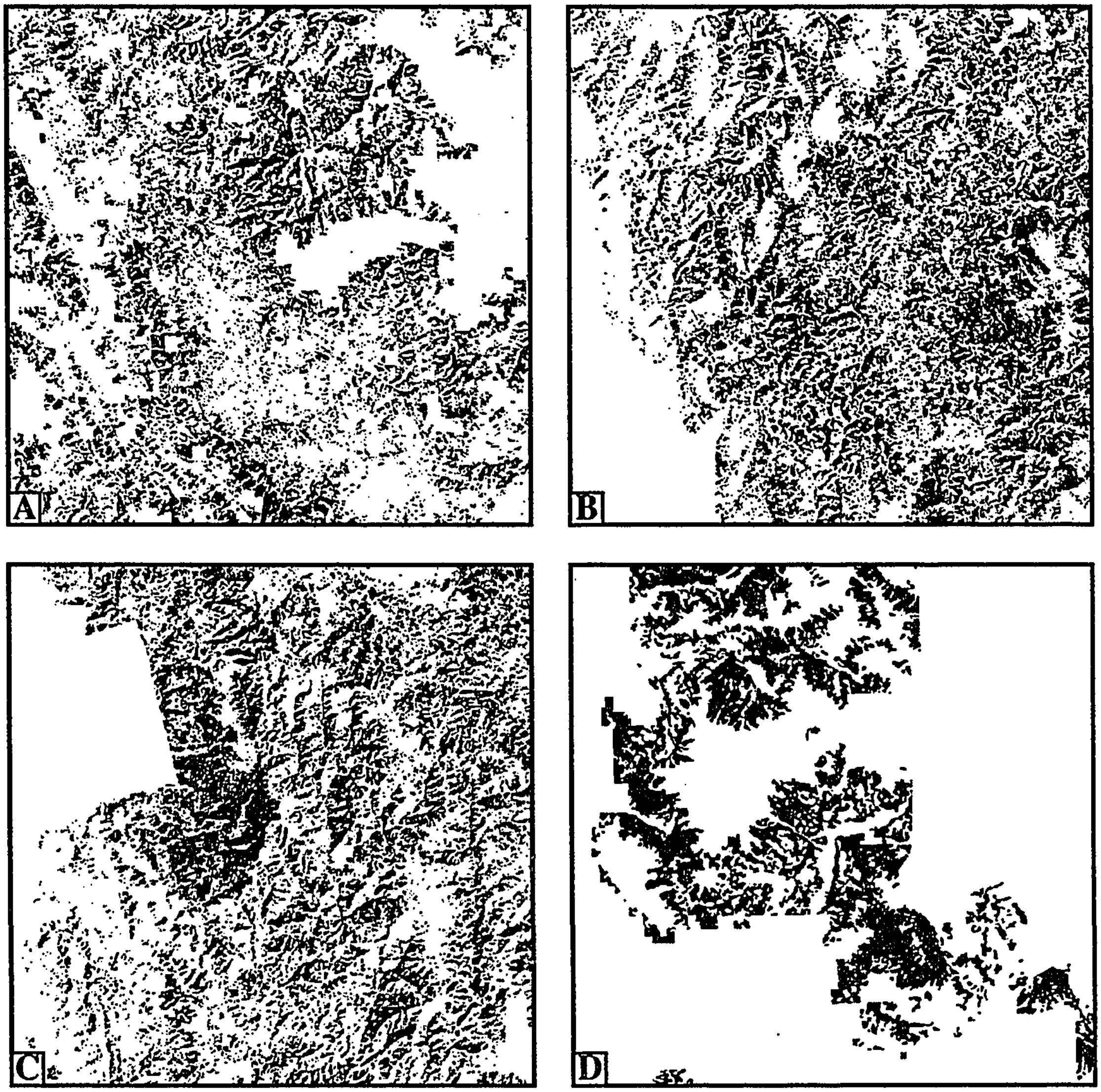


Figure 2. Four of the 1000×1000 pixel sample landscapes used in the analysis. These landscapes represent between 15% and 22% habitat.

necessary to conclude that the landscape figure 2D represents a greater degree of habitat aggregation than figure 2A, the difference between figures 2B and 2C is less obvious.

Pattern indices

All but one of the pattern indices that I examined was computed on a patch by patch basis and then averaged. I defined a patch to be any collection of pixels representing old-growth that touched either at sides or corners. The model assumes that pixels are square, and patch perimeters and areas were thus measured as *(number of pixel edges)* × s and *(number of pixels)* × s^2 where s is the length of a pixel edge. I treated patches that intersected a landscape boundary as if they ended at this edge. A single value for every index of habitat pattern was ascribed to each landscape.

I examined nine commonly used indices of habitat pattern from the landscape ecology literature: number of patches, patch area, core area, patch perimeter, nearest neighbor distance, contagion, perimeter-area ratio, shape index, and fractal dimension. Patch area, core area, perimeter, and nearest neighbor distance were summed over all patches within a landscape. The core area of a patch consisted of all pixels that were at least a single pixel from the patch edge in all directions, including diagonals. I measured nearest neighbor distances from edge to edge. I report perimeter-area ratio, shape index, and fractal dimension as arithmetic, harmonic, and geometric means, as well as means weighted by patch area. The mean-weighted value of an index was computed as

$$\frac{\sum (\textit{pattern index} \times \textit{patch area})}{\sum \textit{patch area}}$$

with the sums taken over all patches within a landscape. Perimeter-area ratios were obtained by simply dividing perimeter by area, shape index (Forman and Godron 1986, Patton 1975) was computed as

$$shape\ index = \frac{perimeter}{2\sqrt{\pi \times area}},$$

and I estimated fractal dimension (see D_a in Milne 1991) as

$$fractal\ dimension = \frac{\log\left[area / s^2\right]}{\log\left[perimeter / 4s\right]}.$$

This method of approximating fractal dimension has little relationship to the true value, which is computed by regressing the logarithms of patch perimeters on the logarithms of patch areas (Milne 1988, Milne 1991). Nevertheless, the quantity in the above expression is frequently encountered, and is discussed here for that reason. I computed contagion as in O'Neill et al. (1988), but with the modification of Li and Reynolds (1993):

$$contagion = 2\ln(n) + \sum\sum P_{i,j} \ln(P_{i,j})$$

where n is the number of categories in the GIS data, and $P_{i,j}$ is the total number of times that category i is adjacent to category j, divided by the total number of times category i is adjacent to all other categories, including itself. For my data, n was equal to two. To remain consistent with the definition of a patch employed in this study, my computations of contagion treated pixels touching at corners as being adjacent.

In addition, experimentation with measures of patch geometry beyond the nine landscape indices described above led me to develop one additional index that to my knowledge has not been previously described. This new quantity, hereafter referred to as patch cohesion, is proportional to perimeter-area ratio divided by shape index when both are computed as means weighted by patch area. Forming this quotient and dropping all constant terms gives the dimensional version of this new index:

$$PC_d = \frac{\sum P}{\sum (P\sqrt{A})}$$

where P and A are patch perimeters and areas, respectively, and the sums are again taken over all patches within a landscape. Patch cohesion can be expressed in a dimensionless form as

$$PC = \frac{max(PC_d) - PC_d}{max(PC_d) - min(PC_d)}$$

where $min(PC_d) = \frac{1}{s\sqrt{N}}$, and $max(PC_d) = \frac{1}{s}$. If s, as before, is the length of a pixel edge, and N is the number of pixels in the entire landscape, then for raster data, $min(PC_d)$ is obtained if every pixel is included in a single patch that fills the entire landscape, and $max(PC_d)$ is obtained if only a single patch of one pixel is present. Inserting these values into the dimensionless expression for patch cohesion gives

$$PC = \left[1 - \frac{\sum p}{\sum (p\sqrt{a})}\right]\left[1 - \frac{1}{\sqrt{N}}\right]^{-1}$$

where $a = A / s^2$ and $p = P / s$ are dimensionless quantities equal to the area and edge of a patch in pixels and pixel edges, respectively. Defined this way, patch cohesion takes only values between zero and one. This choice of non-dimensionalization does not alter the measure's correlation with dispersal success.

Territory allocation

As discussed in the methods section, the model described here is part of a more general life history simulator for the northern spotted owl. I elected to divide each sample landscape into a regular array of hexagonal "sites" both to mimic the territorial behavior of the owl, and because doing so simplified the movement algorithm. The discussion that follows frequently reflects this focus on territorial life histories, but it is unlikely that the results presented here would change if the model was generalized to non-territorial species. I used hexagons to represent territories because they form space-filling arrays and their six immediate neighbors are equidistant.

Territory allocation was accomplished by first intersecting the sample landscapes with an array of hexagons having predetermined size. Individual hexagons in the array were designated suitable or unsuitable depending on the amount of old-growth forest that fell within their boundaries. Two parameters require specification in order to run the territory allocation process. First, it is necessary to determine the size of the hexagons. Because the model works with raster GIS data, it must identify the closest approximation to a true hexagon that remains space-filling. I used four different hexagon sizes in this study containing 36, 360, 672, and 1020 pixels, which for this data correspond to 11.7, 117.0, 218.3, and 331.4 hectares, respectively. The second parameter controlling the allocation of territories defines the minimum amount of old-growth a hexagon must contain to qualify as a suitable site. I refer to hexagons that meet this suitability criteria as territories. I set the suitability threshold at 0.5, which implied that hexagons had to contain at least 50% old-growth to be designated territories. Figure 3 shows the results of running the territory allocation process on the landscape of figure 2C, for 360 pixel hexagons. Habitat in the figure has only been displayed within sites that qualified as territories.

The dispersal model

I designed the dispersal model to minimize the number of parameters needing specification, and to maximize its generality. Dispersers were required to both originate from and settle into territories, and I assumed that every disperser's parents occupied its natal site for the duration of each model run. Thus settlement could never occur in a territory from which a disperser started. No more than one individual could be initialized

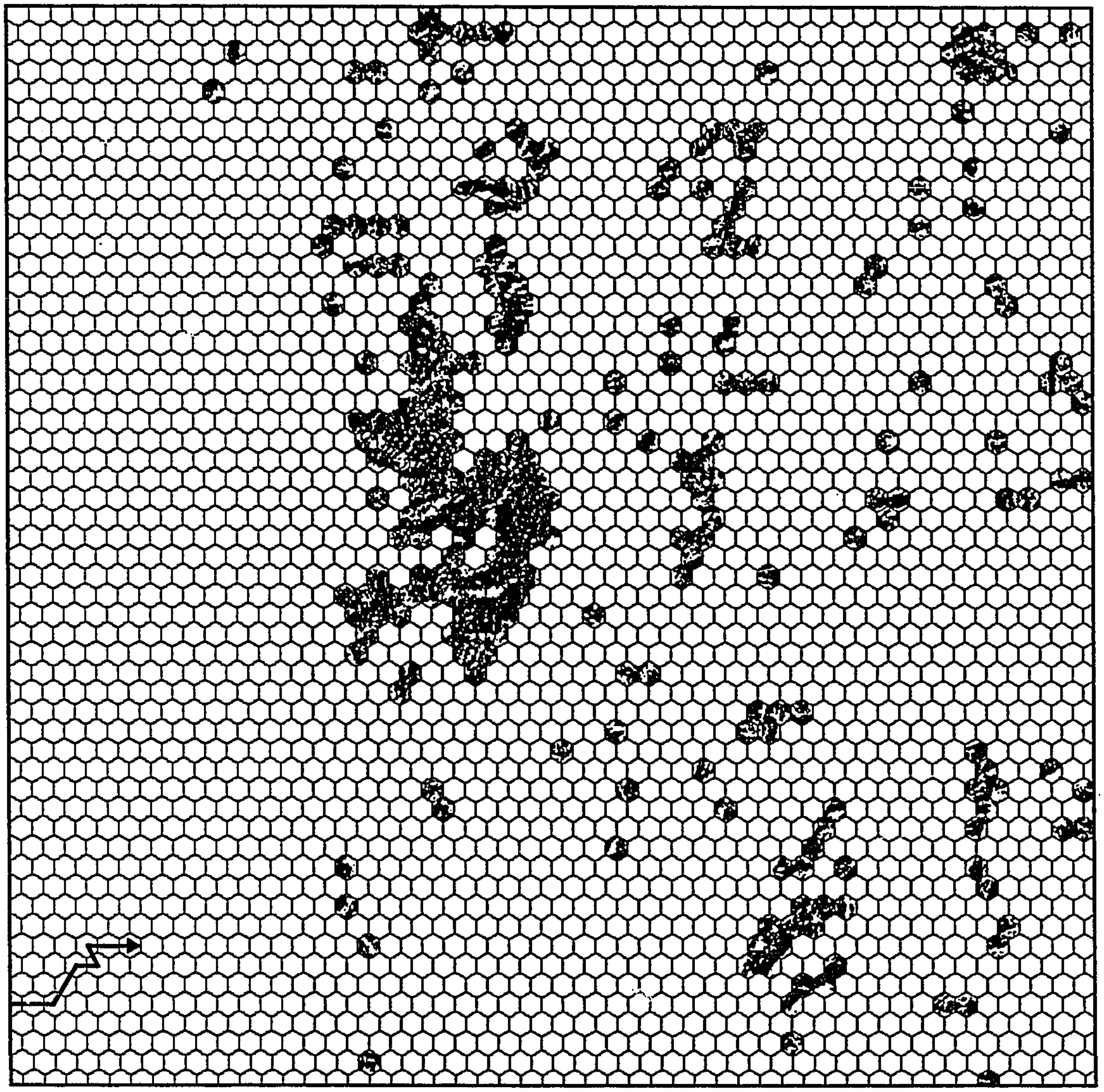


Figure 3. The 1000×1000 pixel landscape of figure 2C following territory allocation. The hexagons are 360 pixels each, and habitat is only displayed for sites qualifying as territories (50% or more old-growth). Eight steps from a hypothetical dispersal path are displayed in the lower left of the figure. Examined from left to right in the direction of the arrow, the path shown first changes direction by moving ahead to the left. It next moves directly ahead, turns ahead to the right, then turns behind to the left, behind to the right, and finally moves directly ahead.

into any given territory, and an unoccupied territory was always settled by the first individual to locate it, at which time it became unavailable to all other dispersers. Dispersers settling into an available territory remained on the site for the duration of the model run. In all of the dispersal runs, 50% of the territories were chosen at random to be initialized with dispersers, and every territory not initialized with a disperser was then available to be settled.

Dispersal paths consisted of a series of individual steps from a given hexagon to one of its six neighbors. In a manner similar to that employed by Stamps et al. (1987), I controlled the overall distance a disperser traveled by imposing a fixed probability of stopping at every step, for each individual, which resulted in path lengths obeying a geometric distribution. The model was parameterized with a value for the mean dispersal path length, which was then automatically converted to the corresponding stopping probability. I refer to individuals that stopped moving before locating a territory as having died. Each living disperser still in search of a territory moved once per time step, and the order in which these individuals moved was randomized at the start of each new time step. Dispersers continued searching until they either died or located an unoccupied territory, and the mean path length, coupled with the number and orientation of territories, governed the disperser's probability of dying. Dispersing individuals could move into an occupied territory, but could not settle in an occupied territory. Dispersal success rate was computed as the fraction of individuals that located territories, and was averaged over a series of separate trials. A reflecting boundary condition was imposed at the edges of the landscapes.

I further specified the model's behavior by controlling the degree of randomness in the disperser's movements. My dispersers were never allowed to remain stationary, and thus every living individual was obliged to move from its current location into one of the six adjacent hexagons each time step. A disperser's choice of movements was not affected by habitat quality or the presence of other individuals. With reference to the previous move, a given disperser could next travel directly ahead, directly behind, ahead and to the left or right, or behind and to the left or right (figure 3). I made the probabilities of choosing each of these directions a function of a single user-defined linearity parameter. If the movement selected was not directly ahead or directly behind, then an additional decision had to be made as to whether it would be to the right or left. In such cases, the probabilities of selecting right vs. left were always equal. The actual probabilities that governed movement were

$$P(ALR) = \frac{(1-z)(1+z)^3}{3}$$

$$P(BLR) = \frac{(1-z)^3(1+z)^2}{3}$$

$$P(DB) = \frac{(1-z)^6(1+z)}{6}$$

$$P(DA) = 1 - \frac{(1-z)(1+z)^3}{3} - \frac{(1-z)^3(1+z)^2}{3} - \frac{(1-z)^6(1+z)}{6}$$

where $z \in [0, 1]$ is the user-defined linearity parameter and *P*(*ALR*), *P*(*BLR*), *P*(*DB*), and *P*(*DA*) refer respectively to the probabilities of moving ahead and to the left or right, behind and to the left or right, directly behind, and directly ahead. I selected these functions because together they allow the single linearity parameter to smoothly control the degree of randomness in the disperser's movements. For this study, I always set the linearity parameter to 0.5.

An inverse relationship exists between the variance in my model's predictions of dispersal success and the number of territories present in a landscape. To control for this variance without needlessly slowing all of the simulations, I altered the number of dispersal trials conducted for a given landscape based on the number of territories it possessed. For landscapes with 100 or less territories, 1000 dispersal trials were averaged to evaluate dispersal success rate. In landscapes with 101 to 250 territories, the model was run 500 times, and in all other landscapes the model was run 100 times. Dispersal runs that involved the 36 or 360 pixel hexagons were conducted within the 1000×1000 pixel sample landscapes. The 2000×2000 pixel landscapes were used for the dispersal trials involving 672 and 1020 pixel hexagons. The larger landscapes were used with the larger hexagons to lower the rate at which dispersers reflected off landscape boundaries, and to increase the absolute number of 672 and 1020 pixel hexagons present in a landscape. The 1000×1000 pixel landscapes contained a total of 27,972 of the 36 pixel hexagons and 2828 of the 360 pixel hexagons (see figure 3). The 2000×2000 pixel landscapes held 6048 of the 672 pixel landscapes, and could fit 3986 of the 1020 pixel hexagons.

Results

Initial tests with all indices

I conducted my initial investigations with 100 randomly selected 1000×1000 pixel landscapes, each containing at least 1% habitat. I computed nine common pattern indices for each landscape plus patch cohesion (see methods). I then ran the dispersal model within each landscape using both 36 and 360 pixel hexagons. I measured the suitability of the pattern indices by examining the product-moment correlation coefficients relating each landscape index to dispersal success (figures 4 & 5). The best predictors of dispersal success were core area, shape index and perimeter-area ratio when weighted by patch area, and patch cohesion. Of the indices I've examined, fractal dimension may be the most frequently reported index of landscape quality, yet it was the index most poorly correlated with dispersal success rate. Core area is a problematic index because it requires specification of an edge width, and no single value is appropriate for all circumstances (Chen 1991, Chen et al. 1992). Correlations of dispersal success rate with the mean-weighted values of perimeter-area ratio and shape index were never as good as those with

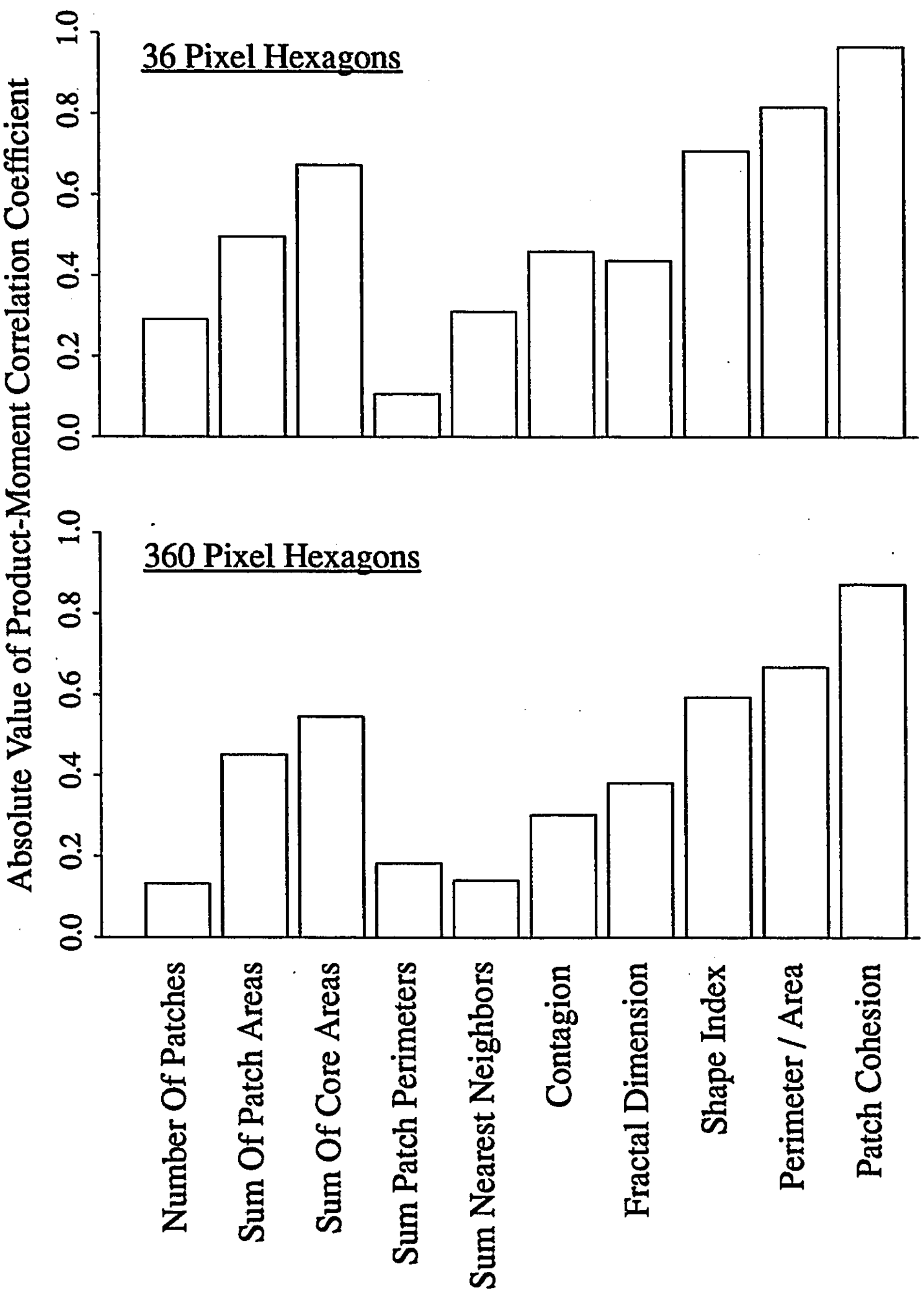


Figure 4. Absolute values of product-moment correlation coefficients relating ten pattern indices to dispersal success rate for both 36 and 360 pixel hexagons. The values for fractal dimension, shape index, and perimeter-area ratio are area-weighted means. See the methods section for a detailed explanation of the pattern indices.

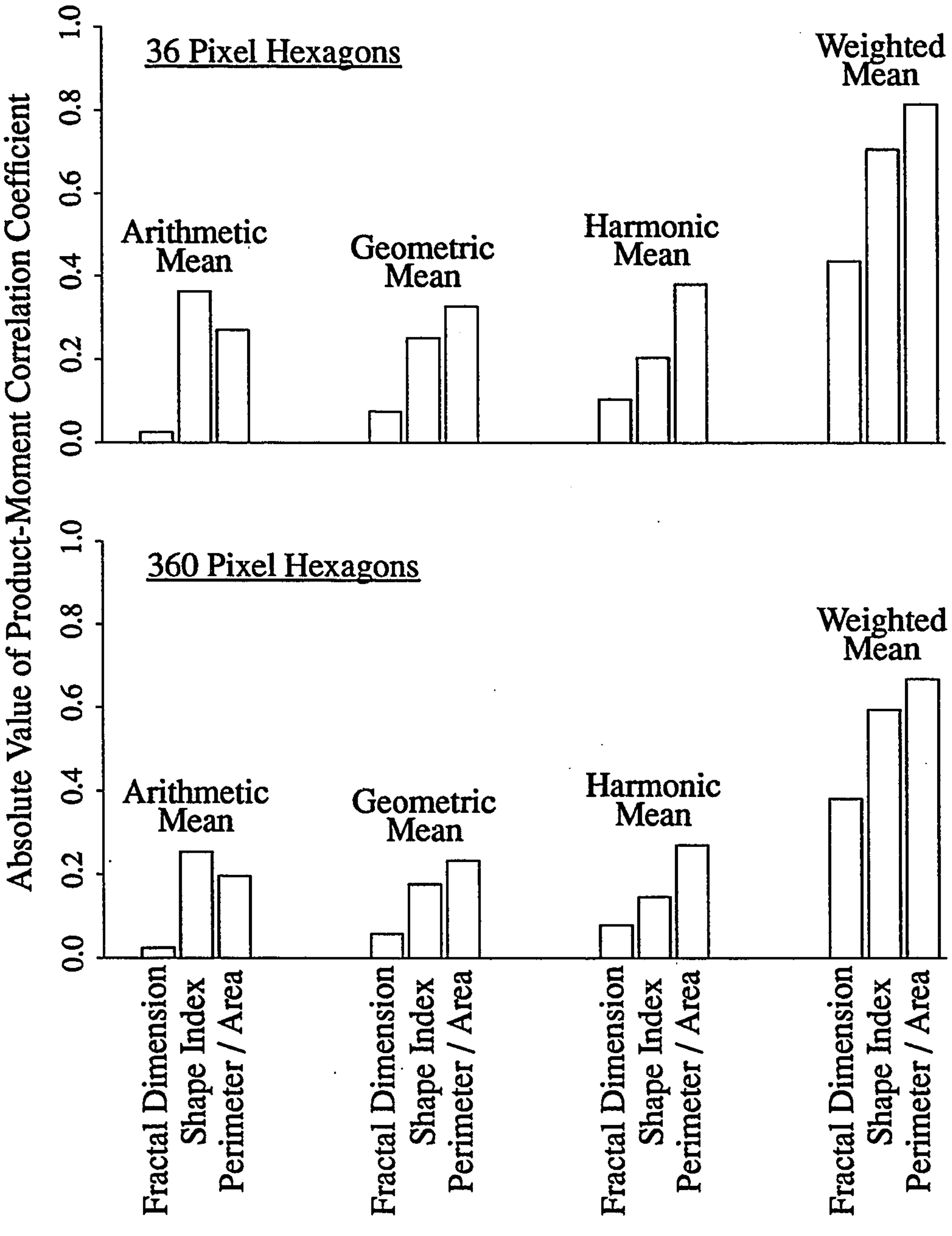


Figure 5. Absolute values of product-moment correlation coefficients relating fractal dimension, shape index, and perimeter-area ratio to dispersal success rate for both 36 and 360 pixel hexagons. The arithmetic, geometric, harmonic, and area-weighted mean values are displayed for each pattern index. See the methods section for the formulation of the area-weighted mean values.

patch cohesion. These results persuaded me to limit further investigation to patch cohesion.

Detailed tests with patch cohesion

I performed additional tests on patch cohesion using hexagons with areas of 36, 360, 672, and 1020 pixels. For each hexagon size I randomly selected 500 landscapes having a minimum of 5% old-growth. Figure 6 shows the scatterplots resulting from computing patch cohesion and dispersal success rate for these four hexagon sizes. The ability of patch cohesion to predict dispersal success rate is evidenced by the product-moment correlation coefficients corresponding to these data, which range between 0.913 and 0.945. The pattern of old-growth in the Pacific Northwest national forests is such that every value of patch cohesion I computed was greater than 0.90.

A more powerful way to examine the data of figure 6 is to select each possible combination of two landscapes and tabulate the frequency with which patch cohesion fails as a predictor of dispersal success. I conducted such a test by discretizing observed differences in the degree of patch cohesion using a bin width of 10^{-4}. Each pair of landscapes contributed to the number of failures or successes associated with its corresponding bin depending on whether the landscape with a higher patch cohesion had a higher measured dispersal success rate. Taking every possible pair of 500 landscapes resulted in a total of 124,750 comparisons being made for each of the four hexagon sizes (figure 7). It is apparent from figure 7 that a difference in patch cohesion of approximately 0.02 will guarantee a correct ranking of dispersal success. In fact, landscape pairs separated by patch cohesion values of 0.02 or more were correctly ranked for dispersal success with a probability of 0.996 for 360 pixel hexagons and 0.999 for the other three hexagons sizes. Between 32% and 46% of the comparisons made corresponded to differences in patch cohesion of 0.02 or more. I placed the landscapes of figure 2 in increasing order (A-D) of both patch cohesion and dispersal success rate, and each pair is

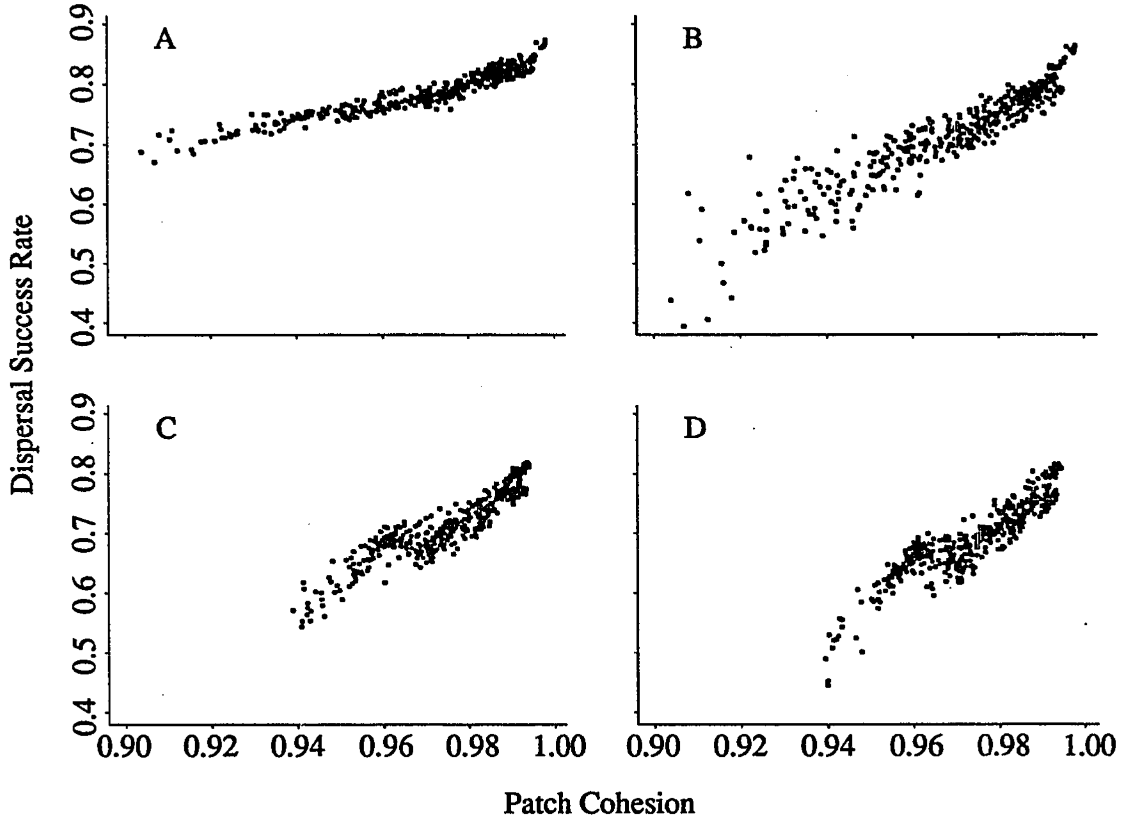


Figure 6. Scatterplots showing dispersal success rate as a function of patch cohesion for each of four hexagons sizes. Plots A, B, C, and D correspond to hexagons of size 36, 360, 672, and 1020 pixels, respectively. Each plot was made by examining 500 landscapes, each containing a minimum of 5% old growth. The landscapes used for plots A and B were 1000×1000 pixels. Plots C and D were made from 2000×2000 pixel landscapes. The product-moment correlation coefficients for plots A, B, C, and D are 0.945, 0.940, 0.920, and 0.913, respectively.

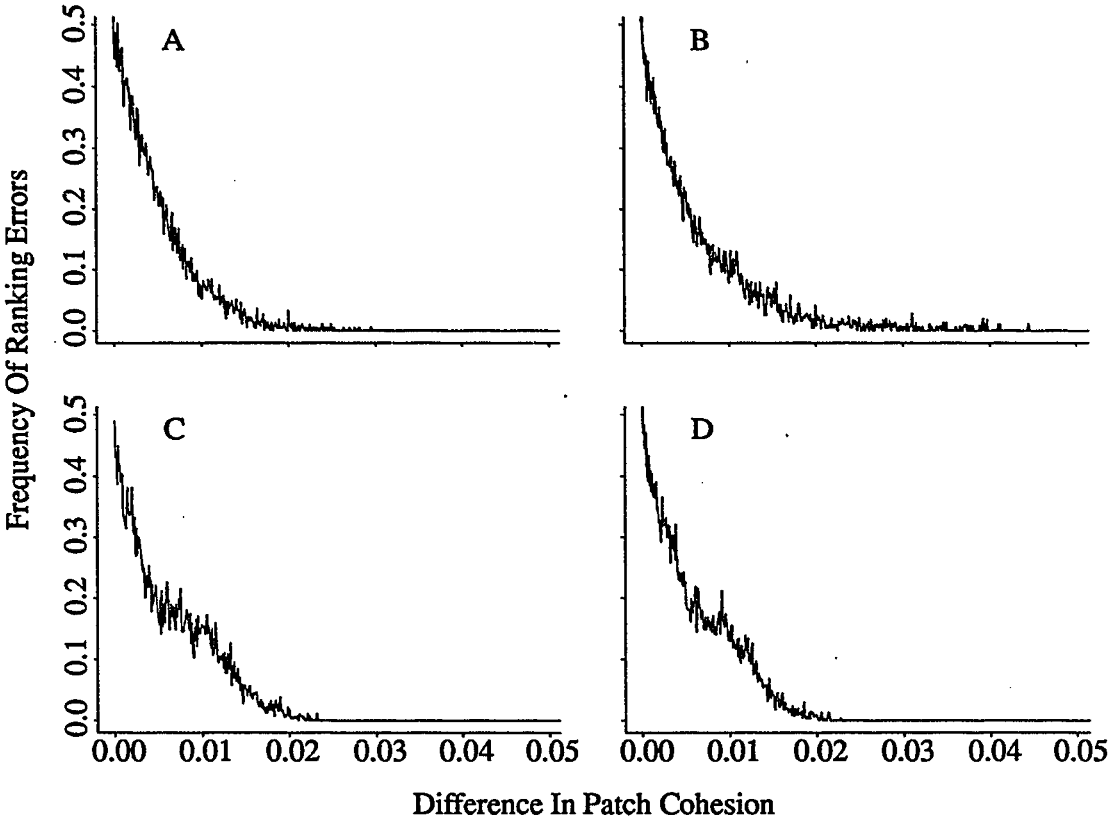


Figure 7. The frequency of committing ranking errors as a function of the separation in patch cohesion for all possible pairings of the data points in figure 6. A ranking error would occur if one member of a pair had a higher value of patch cohesion but a lower dispersal success rate than the other member. Plots A, B, C, and D correspond to hexagons of size 36, 360, 672, and 1020 pixels, respectively. In all four cases, differences in patch cohesion were discretized with a bin width of 10^{-4}, and each plot represents 124,750 separate evaluations. Between 32% and 46% of the comparisons that were made corresponded to change in patch cohesion of 0.02 or more.

separated by a difference in patch cohesion of at least 0.02. The percentage of habitat, degree of patch cohesion, and observed dispersal success rates corresponding to the four landscapes in figure 2 are: (A) 15%, 0.92, 0.58 (B) 22%, 0.95, 0.70 (C) 21%, 0.97, 0.72 (D) 18%, 0.99, 0.80.

I conducted two tests of the sensitivity of my results to changes in the details of the dispersal model. For this analysis, I selected ten of the landscapes from figure 6B that best matched a simple regression line fit to the data. First, I examined the influence of the orientation of the hexagon grid on my results. The exact placement of the hexagon grid on a landscape will affect the number and spatial relationship of territories, and thus the dispersal success rate. However, grid orientation has no effect on a landscape's value of patch cohesion. Thus changing the orientation of the hexagon grid would cause each point in figure 6 to move vertically some unknown amount. I examined how the fit between patch cohesion and dispersal success might change as a function of grid placement by running the territory allocation process for 25 different grid orientations in addition to the model's default. For each of the ten landscapes and each of 26 grid orientations, I ran the dispersal model 1000 times and computed dispersal success rate. Figure 8A shows the range and mean values of dispersal success rate for these ten landscapes resulting from shifting the hexagon grid, as well as the corresponding regression line. The data fit the regression line well ($R^2 = 0.969$) and the equation for the regression line for figure 8A ($slope = 3.527$, $intercept = -2.699$) is almost identical to that of the best fit to the data of figure 6B ($slope = 3.559$, $intercept = -2.732$).

It is clear from figure 7 that landscapes having values of patch cohesion differing by at least 0.02 can be confidently ranked for dispersal success rate. I used the data obtained from shifting the orientation of the hexagon grid (figure 8A) to estimate how much the additional error associated with grid placement will increase the frequency of incorrect ranking of landscapes having this critical separation of patch cohesion. Suppose the "true" values of dispersal success rate are known for two landscapes, that the landscapes are separated by patch cohesion values of 0.02, and that the landscape with the higher dispersal success rate has the higher value of patch cohesion. If they were located on figure 6, these two landscapes would be connected by a line having a slope *m*. In reality dispersal success rates can only be estimated, and these two landscapes might be incorrectly ranked if the total error in the two estimates of dispersal success rate is greater than 0.020 × *m*. Using the slope of the regression line fit to figure 6B for *m*, and assuming the error is shared equally between the two measurements, the maximum tolerable error in a single estimate of dispersal success rate can be assigned a value of 0.035. Model error, estimated as the average displacement from the regression line fit to figure 6B (the regression line is not shown in the figure), is 0.019, which when subtracted from 0.035 yields an estimate of the acceptable error due to grid placement of 0.016. The actual error associated with grid placement can be estimated as the mean displacement from the regression line of figure 8A, which is 0.005, or less than a third of what I have estimated is acceptable. I conclude that the average error associated with grid placement should not cause landscapes to be incorrectly ranked when separated by patch cohesion values of 0.02 or more. This analysis is conservative because in generating the data for figure 6, enough landscapes were selected at random that some pairs ended up only slightly offset from one-another. Such situations produce measurements of dispersal success similar to what would be observed by altering the grid orientation, thus the data in figure 6 already incorporate some variability due to grid placement.

I examined the sensitivity of my results to mean dispersal path length using the same ten landscapes employed for the test of grid orientation. I selected a range of mean dispersal path lengths, and for each landscape and each path length, I conducted 1000 model runs and averaged the results to obtain dispersal success rate (figure 8B). The curves of figure 8B have similar slopes and all are monotonically increasing, which reflects an absence of interaction between the degree of patch cohesion and the change in dispersal success rate associated with altering the mean disperser path length. Thus the strong relationship observed between patch cohesion and dispersal success rate should be invariant to changes in dispersal path length. The value used to obtain the results of figure 6 (25 steps) lies in the middle of the range of path lengths examined here.

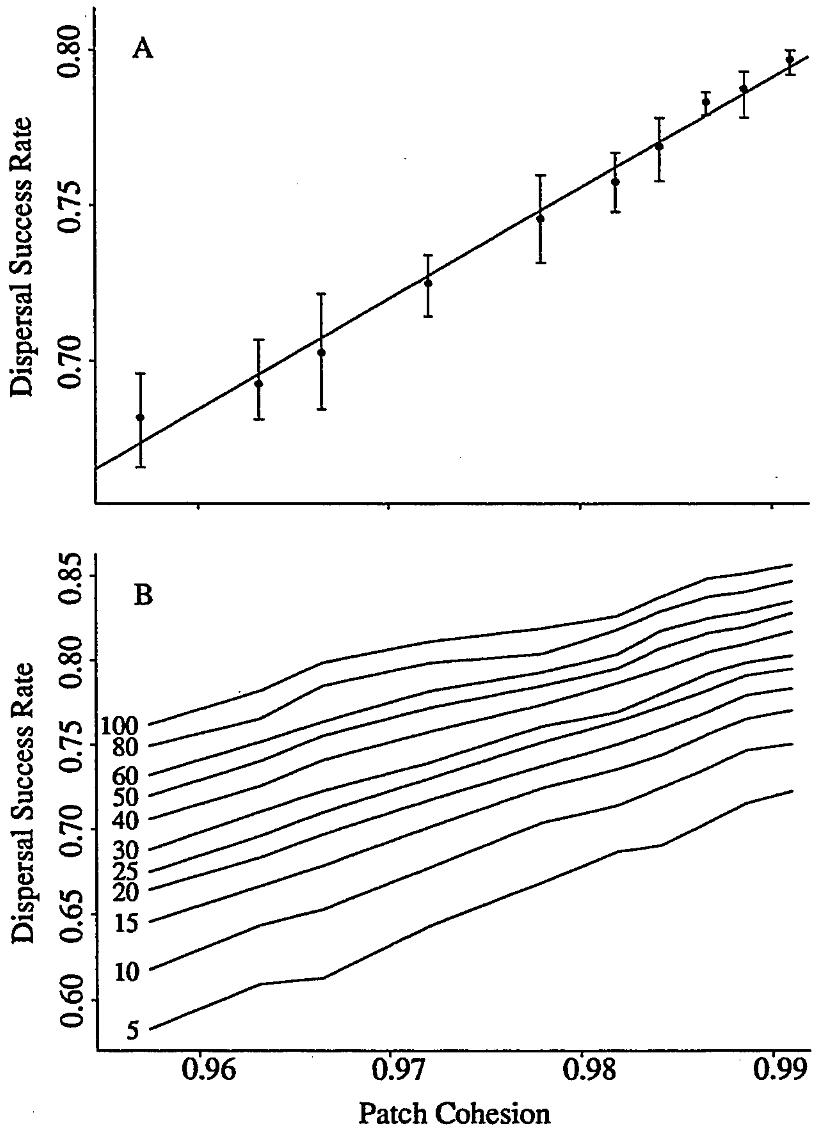


Figure 8. The results from measuring the dispersal model's sensitivity to hexagon grid orientation and mean path length. Both plots were made with ten landscapes that fell on a simple regression line fit to figure 6B. In (A) the dispersal model was run with the default hexagon grid orientation plus 25 additional grid orientations. The figure shows the range of dispersal success rates and the mean values for each landscape. In (B) the mean number of steps taken during the dispersal process was varied. The values of mean path length used in the dispersal model runs are shown to the left of the curves.

To conclude this analysis I used a comparison of real and artificially generated landscape patterns to better illustrate the range of behavior exhibited by patch cohesion. In a manner identical to that described above, I selected 1000 new 1000×1000 pixel landscapes from the full old-growth forest data set. I required that these new landscapes contain at least 1% habitat, and the maximum habitat density I observed in any one landscape was 33.35%. I then constructed 1000 1000×1000 pixel comparison landscapes that uniformly spanned the same range of habitat densities by initializing them into an empty state and then randomly designating individual pixels as habitat until the desired density was achieved. I computed the degree of patch cohesion for each real and random landscape and displayed the results of the analysis in figure 9. For a given habitat density, the real landscapes always exhibited a larger degree of patch cohesion than their artificial counterparts, and the discrepancy became quite large when habitat was sparse. This analysis indicates that, within the range of habitat densities observed in the old-growth forest data, artificially generated landscape patterns may provide little insight into theoretical investigations of animal movement.

Discussion

As a mechanism that lowers habitat connectivity, and thus territory- and mate-finding success rates, fragmentation can cause species' declines in excess of what would be predicted based strictly on estimates of habitat loss (Thomas et al. 1990, Pulliam 1988, Foster and Gaines 1991, Fahrig and Paloheimo 1988*a,b*, Fahrig and Merriam 1985, Saunders et al. 1991, Yahner 1988). Though habitat connectivity can be thought of as inverse to habitat fragmentation, the term has no agreed upon definition (Taylor et al. 1993). To be as meaningful as possible, definitions of habitat connectivity should be predicated on animal movement because animal natural histories are fundamentally linked to landscape pattern through movement. These concerns inspired me to look for numerical relationships between indices of habitat pattern and estimates of dispersal success rate. I examined correlations between nine commonly reported indices of landscape pattern and

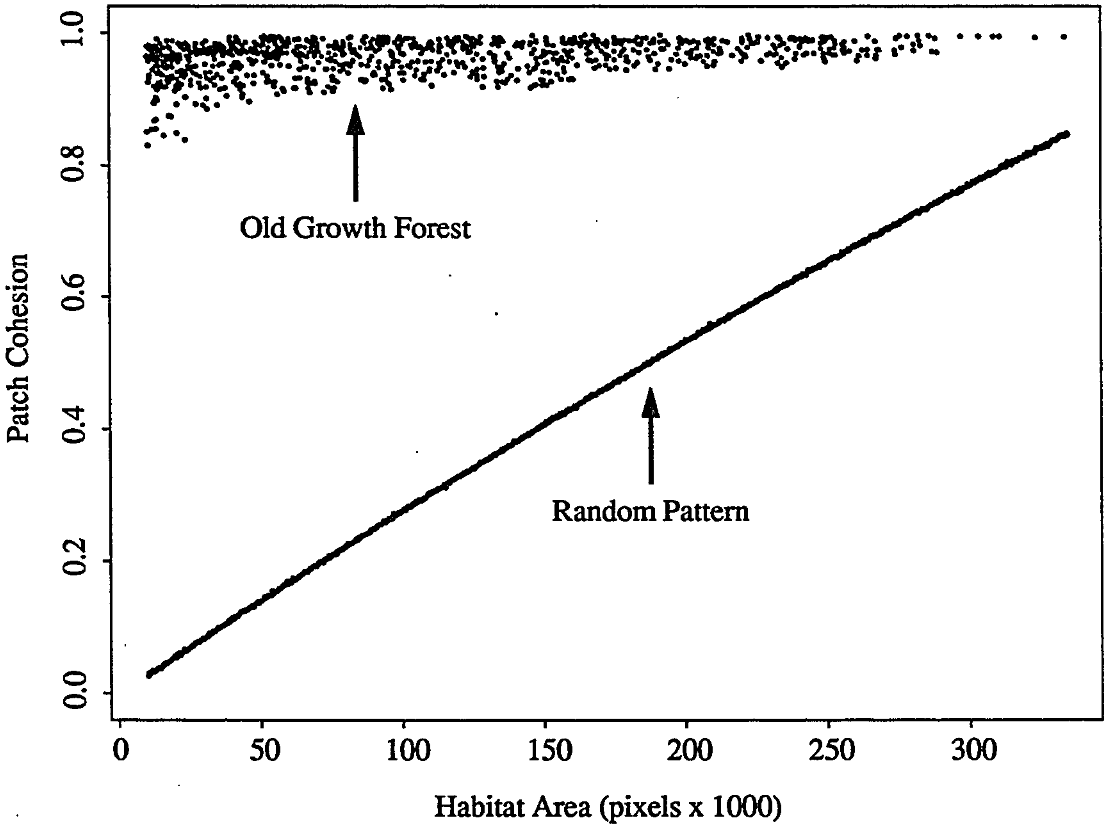


Figure 9. The degree of patch cohesion of 1000 real and 1000 artificial landscapes, plotted as a function of overall habitat area. Each landscape was 1000×1000 pixels in size. The artificial landscapes were obtained by randomly selecting pixels of habitat until the desired density was obtained.

the success of a simulated dispersal process, and found that several frequently cited indices exhibit little predictive power. These weakly correlated landscape indices include fractal dimension, perimeter-area ratio, shape index, nearest neighbor distance, and contagion. Yet I was able to construct a new pattern index that correlated remarkably well with estimates of dispersal success. With the exception of this new index, termed patch cohesion, my results suggest that the pattern indices examined here tell us little about habitat connectivity at large spatial scales. Further tests of these indices could be conducted utilizing response variables other than dispersal success rate, but to the extent that these indices are intended to measure the ecological consequences of habitat fragmentation, the test employed here seems appropriate.

None of the landscapes I examined exhibited patch cohesion values of less than 0.90. This reflects the non-random nature of real landscapes, that result from both ecological and anthropogenic processes, as well as topographic constraints. The landscapes used in this study contain between 1% and 33% old-growth, and as evidenced by figure 9, artificial landscapes containing equal amounts of habitat exhibit dramatically lower degrees of patch cohesion. The small range of values for patch cohesion exhibited by real data is indicative of the dissimilarity between natural landscapes and computer-generated analogues fabricated using simple rules or probabilities. That this discrepancy exists is also apparent from the observation that the expression for patch cohesion is in no way a function of the distances between clusters of habitat. If just the locations of patches within a landscape were altered, then dispersal success rate would change while patch cohesion remained fixed. In this way artificial landscapes could be created for which the predictive power of patch cohesion was greatly compromised. But the landscapes I examined displayed tremendous variation in pattern, and high correlations between patch cohesion and dispersal success rate would not have been observed if they suffered from these pathologies associated with their artificial counterparts. These simple observations suggest that the use of computer-generated landscapes could both inflate the value of poor predictors of ecological quality and diminish the power of useful indices. Yet artificial landscapes have played a substantial role in the development of indices of landscape pattern (e.g. Turner et al. 1989*a,b*, Turner 1990, Gustafson and Parker 1992, Plotnick et al. 1993, Li and Reynolds 1993, but see Houston et al. 1986 for contrast), and the results of this study suggest that pattern indices should be subject to additional scrutiny before being accepted as proxies for true ecological value.

My tests of patch cohesion demonstrate that this index is robust to the details and artifacts of the dispersal model, and this information provides some assurance that other investigators might confidently apply these results to different systems. I measured the frequencies with which patch cohesion failed as a predictor of dispersal success and found that landscapes separated by values of this index of at least 0.02 could be ranked for dispersal success rate with almost no error. Although this analysis was designed for territorial organisms, I suspect the results presented here would not change significantly if the model was generalized to species without territories. This assertion is supported by the observation that patch cohesion correlated strongly with dispersal success over a broad range of territory sizes and dispersal abilities.

Pattern indices that were mathematically similar to patch cohesion were typically only weakly correlated with dispersal success rate (personal observation), even though convincing intuitive arguments could be made for their utility. The poor performance of these pattern indices, and the others examined in this study, argues that the manner in which a simple ecological model integrates spatial information can be lost on even the most appealing indices of habitat geometry. A growing interest in remotely sensed data and GIS technology has resulted in a proliferation of indices of landscape pattern, and the promise of providing powerful tools for communicating complex ecological insights. But if inadequate attention to the complexities of real spatial pattern results in the use of inappropriate landscape descriptors, then more harm may result from their use than good.

Chapter 2: Spotted Owl Population Trends On Washington's Olympic Peninsula

Introduction

Field studies provide the foundation on which ecologists make predictions about how individual species or communities will respond to changes in their environments. Field data, however, are frequently insufficient to allow reliable estimates to be made of population trends into the future. In such situations, a model is typically employed to help make projections from existing data (Doak et al. 1992, Kareiva 1990). Studies of population trends of the northern spotted owl (*Strix occidentalis caurina*) fall into this category. Although, relative to most threatened or endangered species, a great deal is known about the biology of the spotted owl, the available data does not provide an unambiguous picture of the future of the species, especially under differing scenarios of habitat change through time (Holthausen 1994). The research on the northern spotted owl described here was conducted using a model designed to estimate future population trends, and to examine the variation in these trends across space. This model is a two-sex spatially explicit individual-based life history simulator.

Several demographic models for the spotted owl have been developed. Initial efforts were aimed at documenting the possibility of extinction in the foreseeable future, and at conducting sensitivity analysis. These studies began with simple Leslie matrix models (Marcot and Holthausen 1987, USDA 1988), but quickly became more sophisticated through the addition of density dependence, stage-structure, simple feedbacks to spatial pattern, and other biological enhancements (Boyce 1987, Lande 1987, Lande 1988, Doak 1989, Lamberson et al. 1992). After the 1990 listing of the northern spotted owl as a threatened species by the US Fish and Wildlife Service (USDI 1990), the focus of modeling shifted to investigations of the amount and pattern of habitat that might ensure preservation of the species. This new objective necessitated the development of true spatially explicit models that could carefully relate habitat pattern and quality to owl biology. Three such models for the spotted owl have been constructed to date and include that of McKelvey (Thomas et al. 1990, McKelvey et al. 1993), a model by Bart (in prep.) and the work described here. The model employed in this study is similar to that of McKelvey, but the two models were developed independently and differ in many significant ways.

I report here on a study of the possible future of the spotted owl on the Olympic Peninsula of Washington State, USA. The projections of spotted owl population trends described below were obtained using the simulation model I constructed, along with geographical information system (GIS) data compiled by personnel from the Washington Department of Natural Resources (DNR). I describe this GIS data, and present a detailed discussion of the simulation model. Following the model description, I discuss its parameterization using the available data for spotted owls on the Olympic Peninsula. The parameterized model was used to conduct two studies. Initially, I examined the ability of a monitoring scheme that sub-samples a landscape to pick out a region-wide trend in spotted owl pairs. Secondly, I examine the potential contribution of Washington State forest lands to the viability of the spotted owl population on the Olympic Peninsula.

Spotted Owl Biology

The northern spotted owl ranges throughout the Pacific Northwest from northern California to southern British Columbia, and from the Cascade mountains west to the Pacific Ocean. The spotted owl is a long-lived habitat specialist that preferentially selects old-growth forests for nesting, roosting, and foraging. Spotted owls are highly territorial and have home ranges between 1000 and 3500 hectares (USDI 1992). The spotted owl is a monogamous breeder with a low fecundity and clutch sizes ranging from 0 to 3 in any given year (Forsman et al. 1984). Juvenile spotted owls disperse from their natal areas in their first year, and frequently cover long distances in search of a suitable site or mate. Owls lacking sites and mates, typically referred to as floaters, may persist in a landscape for years making seasonal forays into new habitats. The survival and reproductive rates of the northern spotted owl have been strongly correlated to habitat quality (Bart and Forsman 1992, Bart in press). In addition, the survival of dispersing juvenile owls becomes greatly reduced when the landscapes they utilize are fragmented (Doak 1989, Thomas et al. 1990, Lamberson et al. 1992). These aspects of the spotted owl's biology cause it to respond strongly to habitat fragmentation. The owl's long life span is thought to impart a significant delay in the population-level response to habitat loss (Thomas et al. 1990). In addition, low reproductive rates and high dispersal mortality make the species particularly susceptible to changes in habitat connectivity (McKelvey et al. 1993).

Methods

Habitat data

The spotted owl simulation model works with raster GIS data in which each pixel has a single attribute, typically specifying the dominant vegetation type at one point in the landscape. The habitat map used in this study covers the entire Olympic Peninsula, and was pieced together from several sources. Data for the northwest portion of the Olympic Peninsula was based on 1991 Landsat Thematic Mapper (LTM) imagery. The pixel resolution was 30 meters re-sampled to 25 meters. Supervised classification of the LTM imagery resulted in GIS data with nine categories: old-growth, large saw, small saw, pole, sapling, mixed conifer, non-forested, water, and cloud/cloud shadow (table 1). GIS data for the remainder of the peninsula was based on 1990 LTM imagery. This GIS data was classified in a different manner than was the 1991 data. Pixels were categorized as late-successional, mid-successional, or early-successional forests, non-forested, water, or clouds. Then, using land ownership and timber harvest history, pixels were reclassified to correspond with the nine categories of the 1991 LTM image. Some forest stands on the eastern and southern Peninsula did not match the classification criteria for any of the nine categories, and for these a new classification termed "mid-seral" was created. The mid-seral stand type has some value as spotted owl habitat, but less value than small saw.

Table 1: Categories In The Olympic Peninsula GIS Data

Habitat Type	Description
Old-growth	Dominant DBH of 30 inches or greater; usually more than 8 dominant trees per acre; three or more canopy layers with less than complete canopy closure; several snags per acre 20 inches DBH or greater; several down logs per acre 24 inches DBH or greater.
Large saw	Dominant DBH of 20-30 inches with more than 10 dominant trees per acre of this size; co-dominant trees are 14 inches DBH or greater; two or three canopy layers more closed than old growth; small snags present with sparse or no large snags; few large downed logs.
Small saw	Dominant DBH of 14-20 inches; one or two canopy layers; only small snags (or none) present; only small down dead wood (or none) present.
Pole	Dominant DBH 10-14 inches; a single canopy layer; little or no downed dead woody debris.
Sapling	Approximately 2-5 inches DBH.
Mixed conifer	Canopy closure less than 60%; any mixture of at least 90% conifers.
Non-forested	Clear cuts, very young forest, pasture, human settlement.

The GIS data covering different portions of the Peninsula were merged, and an elevation model based on the environmental zones described in Henderson et al. (1989) was used to reclassify forests above 3000 feet in the western and southern Olympic Mountains, above 4000 feet near the middle of the range, and above 4500 feet in the northeastern portion of the mountain range as high elevation forest. In the spotted owl simulations, high elevation forests do not serve as nesting, roosting, or foraging habitat.

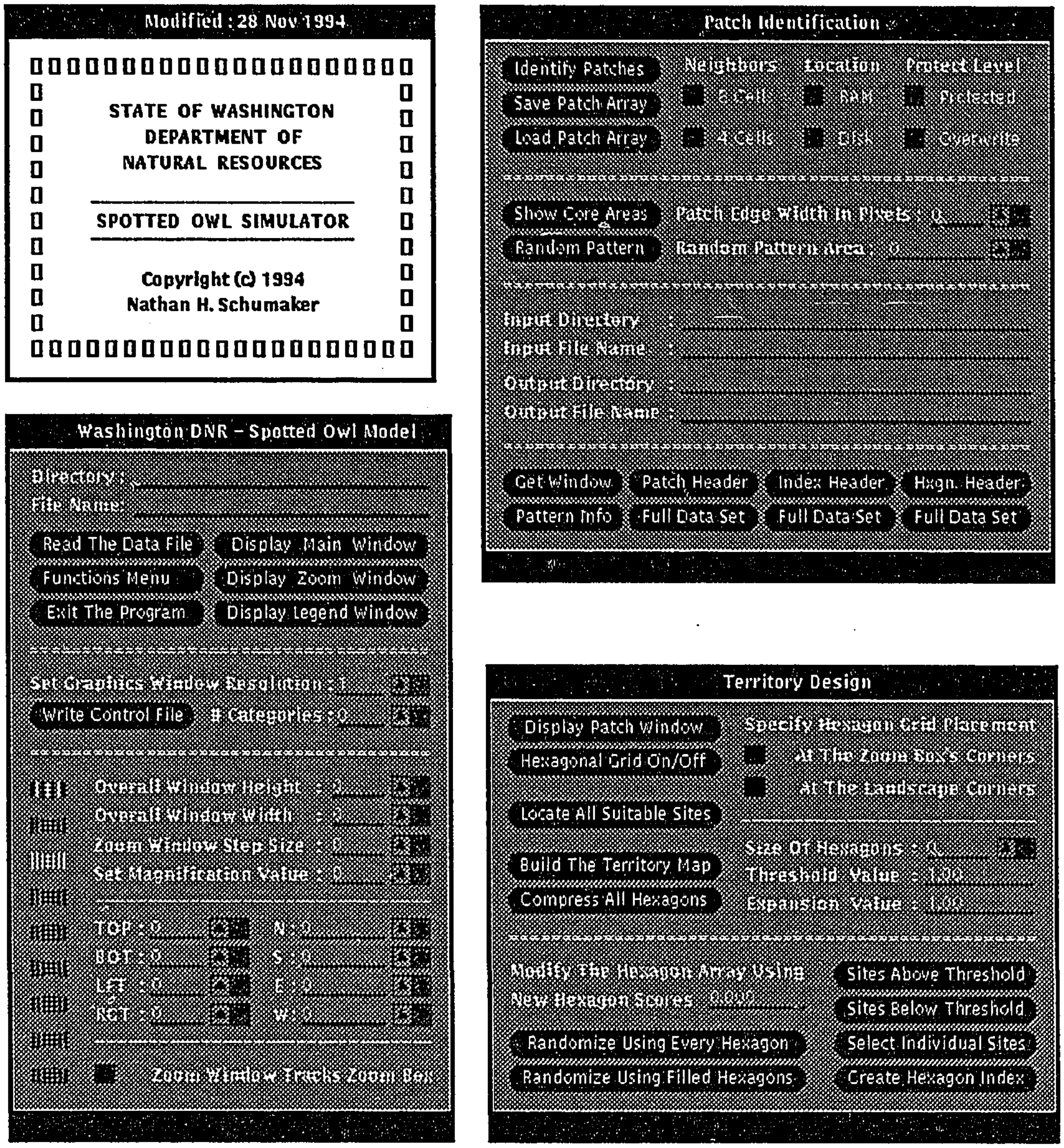
Model overview

The computer model described here was written in the C programming language and runs on a SUN Workstation^{TM}. The model is designed to be used with raster GIS data showing the distribution of habitat across space, and includes separate modules that conduct territory allocation, movement modeling, and spotted owl simulations (figures 10 and 11). The territory allocation module is used to generate a new GIS layer that specifies the locations and qualities of territory-sized units of habitat. The movement module controls both the dispersal of fledglings and seasonal searching of floater owls. The spotted owl simulator tracks the survival, movement, and reproduction of an entire population of spotted owls on an individual basis. These routines are all housed within a graphical user interface, and the processing and display of GIS data is conducted within the model itself. The movement and spotted owl sub-models were animated to facilitate the parameterization process.

The spotted owl simulator is designed to make several life history parameters functions of habitat quality. These functional relationships are accomplished through the assignment of relative weighting values to the classes of habitat present in the GIS data. Spotted owls in the model interact with the landscape at the scale of an individual territory, and the relative weighting values assigned to the GIS data categories influence the owls by controlling the quality of these territory-sized portions of the landscape. An owl in a superior site will experience a higher average survival rate, and is more likely to remain on the site and breed, than an owl in an inferior site.

The territory allocation process

A map identifying clusters of habitat capable of functioning as spotted owl territories is generated through the intersection of the GIS habitat data and a regular array of hexagons. Throughout the text, I refer to hexagons as sites, and sites containing enough habitat to



Set Graphics Window Resolution : 1

\# Categories : 0

Overall Window Height : 0
Overall Window Width : 0
Zoom Window Step Size : 0
Set Magnification Value : 0

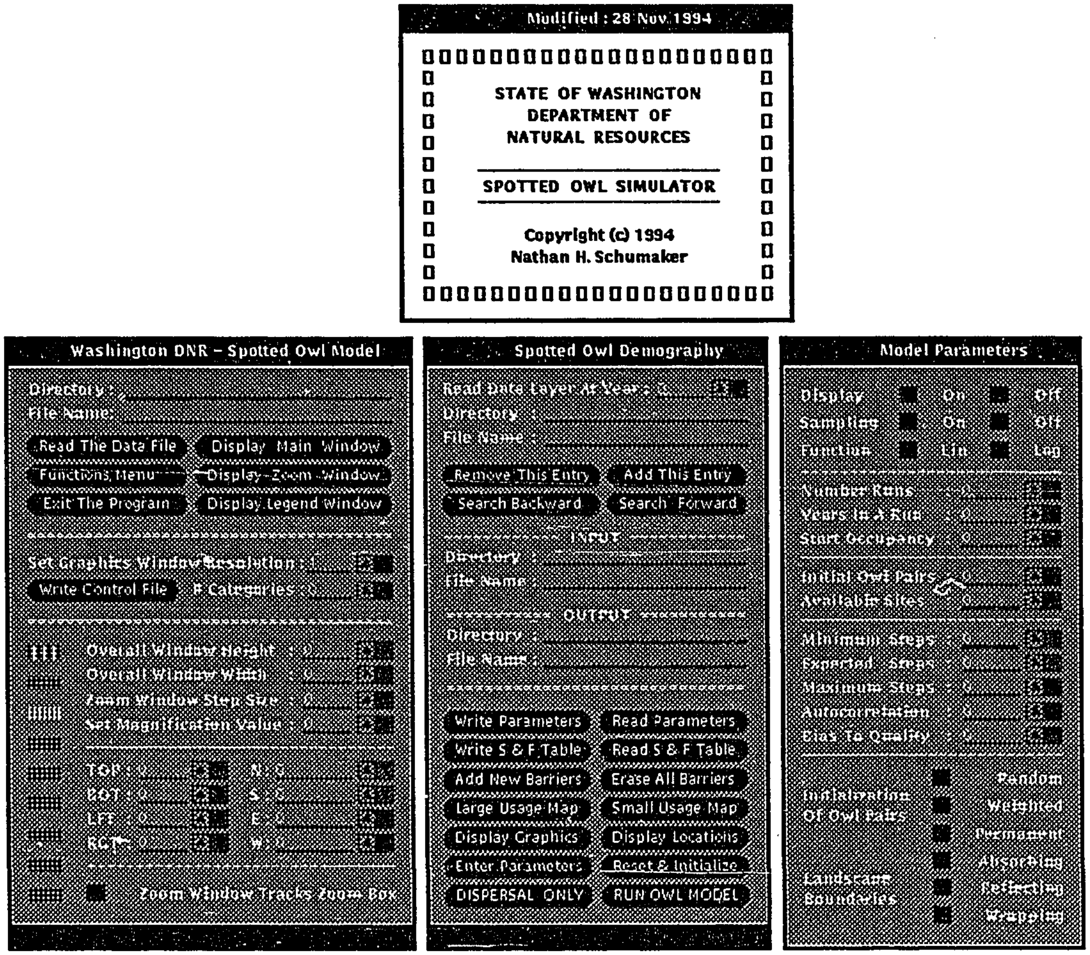
TOP : 0
N : 0
BOT : 0
S : 0
LFT : 0
E : 0
RGT : 0
W : 0

Patch Edge Width In Pixels : 0
Random Pattern Area : 0

Size Of Hexagons : 0
Threshold Value : 1.00
Expansion Value : 1.00

New Hexagon Scores 0.000

Figure 10. The simulation model. This figure shows the main window as well as the patch identification and territory allocation modules. The model's graphics windows are not shown in the figure.



Modified : 28 Nov 1994

STATE OF WASHINGTON
DEPARTMENT OF
NATURAL RESOURCES

SPOTTED OWL SIMULATOR

Copyright (c) 1994
Nathan H. Schumaker

Figure 11. The simulation model. This figure shows the main window and the spotted owl simulation module. The model's graphics windows are not shown in the figure.

support a breeding pair of owls are called territories. No distinction is intended between references to territories and to owl home ranges. The distribution of habitat within the GIS data, the relative weights assigned to the habitat categories, and three additional model parameters control the distribution of territories across a landscape. The three additional parameters specify hexagon size, the amount of habitat necessary to support a breeding pair of owls, and the maximum amount that any two territories can overlap. The territory allocation module uses these parameters and the relative habitat weights to convert the GIS habitat map into a new map that displays the locations of suitable breeding sites across a landscape (figure 12).

A consequence of the use of raster GIS data is that individual sites in the model are actually raster approximations of true hexagons. The model is designed to ensure that every pixel in the GIS data set is allocated to one and only one hexagon, and that every hexagon is identical in shape and contains the same number of pixels. When the hexagon size is increased or decreased, the model increments or decrements each hexagon's width by two pixels (symmetry would be lost if hexagon widths were altered by a single pixel). Once the width has been set, the mathematical relations that define a true hexagon are used to identify those data pixels that fall inside a specific model hexagon. A score for each site is obtained by summing the relative weights of each of its pixels, and then dividing this sum by the total number of pixels in a hexagon. This score has a value in the range specified by the lowest and highest weights assigned to a habitat category. A single model parameter sets the threshold score a site must have in order to be designated suitable for breeding. Owls in the model are influenced solely by these two attributes associated with each hexagon (its score and its designation as a territory or non-territory).

The mechanism for territory delineation described above systematically underestimates the number of home ranges a landscape is capable of supporting because sites below the threshold go unused even if sufficient habitat is near. To correct for this biologically unrealistic artifact of the regular hexagonal grid, I designed the model to allow habitat to be shared across hexagon boundaries. This sharing process is controlled through a single

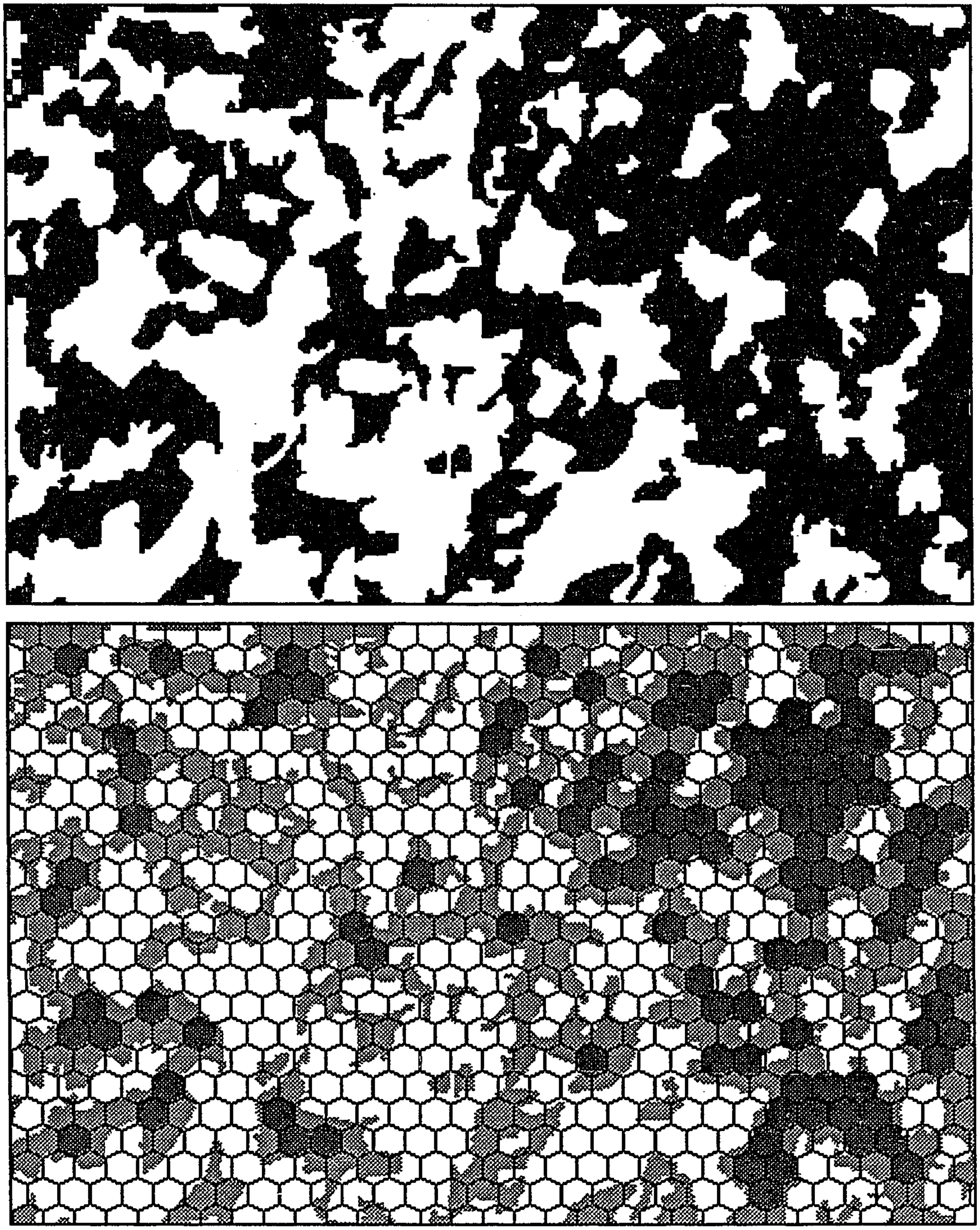


Figure 12. The territory allocation process. The upper panel shows habitat in black. The lower panel displays the same map after territory allocation. Suitable sites are dark gray in the territory map. This small example was not taken from an actual simulation run.

"territory expansion" parameter. At this parameter's lower limit, no sharing takes place, while at its upper limit a hexagon may make use of all of the "available" habitat contained within its six immediate neighbors. If a neighbor's score falls below the suitability threshold, then all of its habitat will be available for borrowing. A neighbor with a score greater than the threshold may lend only this excess. The sharing of habitat takes place beginning with sites closest to the threshold value, and habitat is taken initially from neighbors having the greatest surplus. To the extent possible given these constraints, any fraction of the habitat contained within a hexagon's six neighbors may be shared (up to the limit specified by the territory expansion parameter). Hexagons that expand into neighboring habitat take only what is necessary to bring their score up to the threshold value that defines a site as suitable. Habitat borrowed by one hexagon is unavailable for use by any other hexagon.

Hexagon boundaries and scores do not change as a result of territory expansion. Instead, this mechanism simply considers the presence of nearby habitat when sites are designated as suitable for breeding. Borrowed habitat is not included in the computation of a site's score, so a hexagon that has become suitable through borrowing will have a lower score, and therefore function as a lower quality site than one containing exactly the threshold amount of habitat. While in some situations it may seem appropriate to increase a hexagon's score to reflect borrowed habitat, retention of the original low score is intended to reflect an increased cost of foraging within and defense of a larger territory.

Movement modeling

The movement model controls the searching process of both floater and fledgling spotted owls, and functions identically for both except that fledgling owls can be required to move a minimum distance before settling, and this is not applied to floaters. In the model, male owls always move first, and search for available suitable sites in which to set up territories. Female owls move after the males and search for single territorial males to pair with. In the discussion below, I use the term “available site” to refer to both an available suitable site, in the case of a male, and in the case of a female, a single territorial male. There is no mortality function within the movement model, and owls that stop moving before locating an available site simply become floaters. In reality, spotted owls actually incur significant mortality during dispersal (Gutiérrez et al. 1985, Thomas et al 1990 Bart and Forsman 1992), but the available data is not adequate to confidently separate it from the best estimates of overall mortality (Burnham et al. 1994). Thus all mortality in the model is collapsed into a single survival function. Owls will always locate a suitable site that borders their present location. Movement decisions are not affected by the presence of other individuals, and searching owls can move into, but may not settle in, an occupied territory. The model tracks the fate of each individual, as well as the net and sum distances moved by dispersing fledglings. Net distance refers to the length of a line connecting the starting and ending locations, while sum distance is intended to describe the total path length.

The parameters that control the movement model are the minimum, mean, and maximum number of steps each individual may take, the movement linearity, bias to site quality, and a boundary condition. The minimum number of steps effects only the fledgling owls, and specifies the number of steps that must be taken before an available site may be occupied. The maximum number of steps effects both fledglings and floaters, and any owl that continues to move until this limit is reached will stop and become a floater. Movement paths consist of a series of individual steps taken from a given hexagon to one of its six neighbors. I control the overall distance an individual travels by imposing a fixed probability of stopping at every step, for each individual, which results in path lengths obeying a geometric distribution. The mean path length parameter is automatically converted to the stopping probability that produces a geometric distribution with this mean. Each individual in search of a territory moves one step at a time, and the order in which these steps are taken is continually randomized. Each individual continues to move until an available territory is located, or until a decision to stop is made.

In addition to the distance travelled, I also specify the searching owl's behavior by controlling the degree of randomness in individual movements. Every owl in motion is obliged to take individual steps from its current location to one of the six adjacent hexagons, and the choice of direction may be affected by both the direction of the last movement, and by the neighbor's scores. With reference to the previous move (and as a consequence of a hexagon's shape), a given individual can next travel in one of the following relative directions: directly ahead, directly behind, ahead and to the left or right, or behind and to the left or right (figure 13). I made the probabilities of choosing each of these directions a function of a single user-defined linearity parameter that takes values between zero and one. If the movement selected is not directly ahead or directly behind, then an additional decision must be made regarding whether it will be to the right or left. In such cases, the probabilities of selecting right and left are always equal. When the linearity parameter is set to zero, the choice of direction is random, while at one, a searcher always travels in the previously selected direction. Values of this parameter that fall in-between one and zero make the movement more or less strongly correlated with the direction of the previous step. The actual probabilities that govern these choices of movement direction are:

$$P(ALR) = \frac{(1-z)(1+z)^3}{3}$$

$$P(BLR) = \frac{(1-z)^3(1+z)^2}{3}$$

$$P(DB) = \frac{(1-z)^6(1+z)}{6}$$

$$P(DA) = 1 - \frac{(1-z)(1+z)^3}{3} - \frac{(1-z)^3(1+z)^2}{3} - \frac{(1-z)^6(1+z)}{6}$$

where $z \in [0, 1]$ is the user-defined linearity parameter and $P(ALR)$, $P(BLR)$, $P(DB)$, and

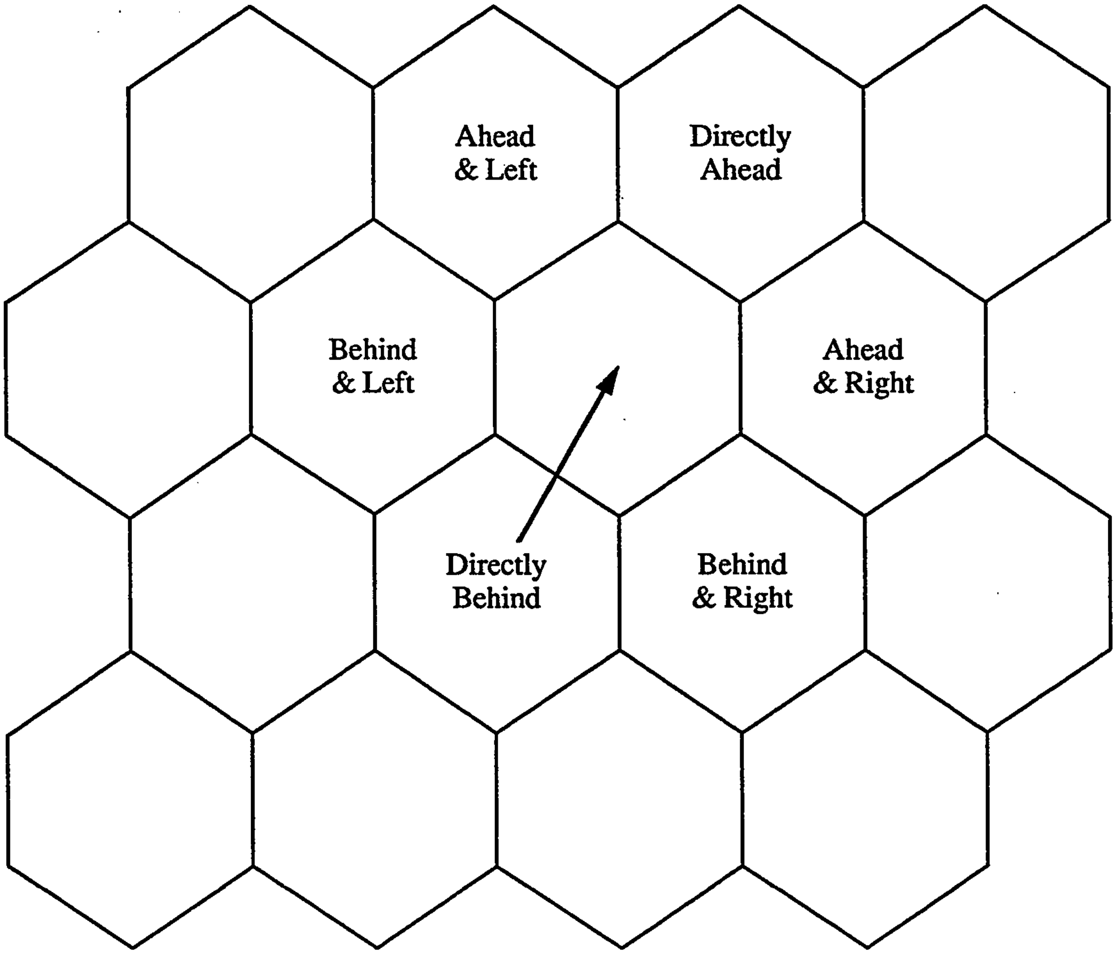


Figure 13. The naming convention used to define relative directions within the movement module. The previous step is indicated by the arrow in the figure, and the labeled directions describe the choices available for the next move.

P(*DA*) refer respectively to the probabilities of moving ahead and to the left or right, behind and to the left or right, directly behind, and directly ahead.

As mentioned above, movement decisions can also be influenced by the quality of the surrounding habitat. This is again accomplished through a single parameter that takes values between zero and one. Movement decisions made based on habitat quality override those decisions which are based on movement linearity. This behavior, termed "bias to site quality" is accomplished by setting a frequency with which movements are simply made to the neighboring site with the highest score. A true random walk can be obtained by setting both the bias to site quality and the movement linearity to zero. Increasing the movement linearity produces a directed random walk, and increasing the bias to site quality has the effect of concentrating the searching effort in areas of superior quality. Lastly, the edges of the GIS data can be made to function as either reflecting, wrapping or absorbing boundaries, and additional reflecting boundaries can be located within the territory map to delineate natural features such as coastlines or mountain ranges.

The spotted owl simulator

The spotted owl simulator is a three stage, individual-based, spatially explicit model that follows both males and females from birth to death. The model has a yearly time step which is broken up into the movement of floaters, reproduction, movement of fledglings, and survival. The model makes a variety of demographic projections including the number of owls in each age class (adults, subadults, and juveniles), the number of pairs, single territorial birds, and floaters, the mean dispersal distance, and mean survival rates. A sampling facility generates a table of data that allows the population in any subset of the landscape to be observed at any time. In addition, the model generates an occupancy map that can be used to investigate the relative usage by owl pairs of various portions of a landscape. A built-in time series function allows the model to read new territory maps on a yearly basis, and thus to simulate landscape change through time. The movements of both floater and fledging owls are controlled by the movement sub-model. The survival and site fidelity processes are stochastic, with decisions made each year on an individual basis. Reproduction, however, is deterministic except for the manner in which fledgling owls are distributed to breeding pairs. Environmental stochasticity can be incorporated into the simulations through a table of survival and fecundity input values. An overview of the flow of events in the spotted owl simulator is shown in figure 14.

To run the spotted owl simulator, it is necessary to assign values to several demographic parameters. A list of the parameters than must be given values appears in table 2. The movement model requires specification of the minimum, mean, and maximum number of steps to be taken by each individual owl. In addition, the movement linearity and bias to site quality control the dispersal behavior of fledglings and the searching behavior of floaters. Survival and fecundity values are placed into a table, and each year the model selects a single row from this table at random. It is possible to introduce environmental stochasticity into the simulations by correlating survival and fecundity across rows. In addition, a distribution of survival and fecundity values can be specified by placing predetermined numbers of identical rows into the input table. The survival and site fidelity functions both require that a minimum and a maximum value be specified, and these define the parameter's values in the worst and best sites. Survival and fidelity decisions taking place in sites that fall in-between the extremes of quality are made by interpolating between these limiting values. This interpolation can be conducted using either a linear or logistic function, depending on the value of a "fit to site quality" parameter. Survival and site fidelity decisions are based strictly on site quality, with each event determined through the selection of a uniformly distributed random number.

The number of new fledglings produced per age class is computed in aggregate, irrespective of site qualities. To this extent, the reproduction process is deterministic. Randomness enters only in the manner in which these fledglings are apportioned out to the breeding pairs. Fecundity was not made a function of site quality because Lotka-Leslie models for the owl are not particularly sensitive to fecundity (Noon and Biles 1990,

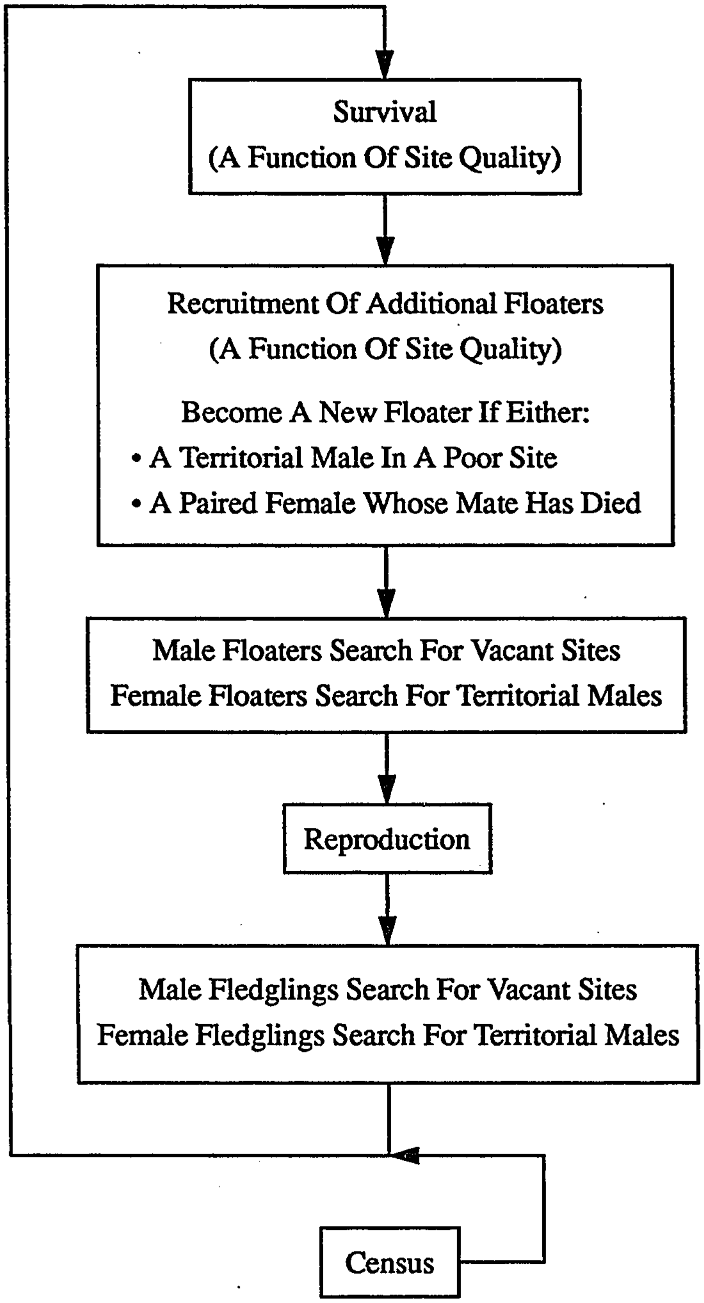


Figure 14. The flow of events that take place within the spotted owl simulator in a single year.

Thomas et al. 1990), and in order to keep the model as simple as possible. A maximum brood size must be specified, as well as the probabilities that a given pair of owls will produce a brood of any specific size. The model does not simulate nest mortality.

Table 2: Parameters In The Simulation Model

Parameter	Description
Minimum number of movement steps	Integer ≥ 0
Mean number of movement steps	Integer > 0
Maximum number of movement steps	Integer > 0
Movement linearity	0 ≤ Integer ≥ 100
Bias to site quality	0 ≤ Integer ≥ 100
Sex ratio	0 < Integer > 100
Maximum brood size	Integer > 0
Brood size probabilities	0 ≤ Integers ≥ 100
Site fidelity minimum and maximum	0 < Integer > 100
Min. & max. juvenile, subadult, and adult survival	0.0 ≤ Real ≥ 1.0
Juvenile, subadult, and adult fecundity	Real ≥ 0.0
Fit to site quality	Linear / Logistic
Initialization method	Random / Weighted / Custom
Boundary condition	Absorb / Reflect / Wrap

The simulation model is initialized with only pairs of adult owls; other stage classes as well as floaters and single territorial birds are generated after the model is started up. The model allows the initial owl pairs to be located randomly, into the best collections of sites, or to be placed (in part or in full) by hand with the computer's mouse. If the initialization of owl pairs is strictly random, and if multiple runs are to be conducted, then it also becomes necessary to specify whether these initial locations will be used in all model runs, or if new initial locations should be picked at the start of each run. Individual sites can be made to function as reflecting boundaries in order to prevent movement across large bodies of water or mountain ranges, etc. In addition to the parameters described above, it is also necessary to produce at least one territory map using the model's territory allocation procedure.

Parameterization of the territory module

The process of territory allocation is governed by weights assigned to the GIS habitat categories, the hexagon size, the suitability threshold, and the territory expansion parameter. The GIS data for the Olympic Peninsula contained 8 forest stand types (table 1). Old-growth, large saw, and small saw are the only stand types that owls regularly use in proportion to their availability (Thomas et al. 1990, Carey et al. 1992). Thus, only these habitat types, and the mid-seral category, were assumed to be of use to the owls in the simulations. In order to assign relative weights to the old-growth, large saw, and small saw stand types, a habitat utilization index (HUI) was calculated for each. A HUI describes habitat preferences within a home range and can be defined as

$$\text{HUI} = \frac{\text{fraction of time spent in habitat type } \tau}{\text{fraction of home range consisting of habitat type } \tau}.$$

A preference for a specific habitat would be expressed as an HUI greater than one, while values less than one would represent avoidance. The relative habitat weights were obtained from the quotients of the HUI values for various habitats, under the assumption that these values reflect the ability of a stand type to meet the habitat needs of a spotted owl. Using data collected from 20 owl home ranges on the western Olympic Peninsula (E. Forsman, unpublished data), the calculated ratios were

$$\frac{\text{HUI}_{OG}}{\text{HUI}_{LS}} = 1.50 \quad ; \quad \frac{\text{HUI}_{OG}}{\text{HUI}_{SS}} = 2.19$$

where the subscripts OG, LS, and SS stand for old-growth, large saw, and small saw, respectively. Assuming that the relative habitat weights, which are integers, should produce the same ratios requires that

$$\mathrm{Wght}_{LS} = \frac{1}{1.5} \cdot \mathrm{Wght}_{OG} \quad ; \quad \mathrm{Wght}_{SS} = \frac{1}{2.19} \cdot \mathrm{Wght}_{OG} .$$

Wght_{OG} was arbitrarily set to the largest possible weight, which is a value of nine. Using the above relations, the values for Wght_{LS} and Wght_{SS} were set to six and four respectively. The mid-seral habitat category was assigned a weight of one because this represents the smallest non-zero weight.

The hexagon size is intended to correspond roughly to the minimum home range size of an owl pair. Hexagon size and the expansion parameter together specify the maximum home range size. The Olympic National Park currently possesses the highest density of good quality owl habitat on the Olympic Peninsula, and spotted owl home ranges vary inversely with the density of high quality habitat (Carey et al. 1992). Thus, the peninsula's smallest spotted owl home ranges are expected in the Olympic National Park. The density of owl pairs in the low-elevation old-growth forests of the park is estimated at 0.08 per 100 hectares (E. Seaman et al., unpublished report) and, assuming territories do not overlap, this density could be obtained with an average home range size of 1250 hectares per pair. Given these considerations, the territory allocation module's hexagon size parameter was set as close to 1250 hectares as possible given the raster nature of the GIS habitat data. The actual hexagon size used in the simulations was 1269 hectares.

The threshold value that a site must have in order to qualify as a territory was computed from the relative weights and the above-mentioned data on 20 western Olympic Peninsula home ranges (E. Forsman, unpublished data). The relationship used was

$$\text{suitability threshold} = \sum \text{weight} \cdot \text{representation}$$

where weight refers to a user-defined habitat weight, representation refers to the mean frequency of occurrence of a habitat type within the spotted owl home ranges sampled, and the sum is taken over all habitat types in the GIS habitat data. This relationship gives a suitability threshold of roughly 5.0. The observed distribution of home range sizes was modeled using the territory expansion parameter, which effectively specifies an upper limit to the size of any territory. The Olympic Peninsula home range data (E. Forsman, unpublished data) gives mean and maximum home ranges of 5788 and 11,056 hectares, respectively. The territory expansion parameter at most allows expansion into a hexagon's six immediate neighbors, and with a hexagon size of 1269 hectares this specifies an upper limit on territory size of 8883 hectares. This value is below that of Forsman's data, and thus the maximum expansion value was used. The territory map that resulted from the use of these parameters includes 1239 sites containing at least some habitat, of which 435 were suitable (figure 15). This map has been reviewed by several spotted owl biologists with extensive field experience on the Olympic Peninsula, and is generally accepted as a plausible depiction of the distribution of potential spotted owl home ranges across the Olympic Peninsula (S. Horton, personal communication).

Parameterization of the spotted owl simulator

Additional parameters that were estimated from field data include the five movement parameters, sex ratio, the brood sizes and their corresponding probabilities, and the survival and reproduction rates. The actual survival rates and movement distances are not fixed in the model, but are functions of both the input parameter values and the quality and spatial arrangement of habitat. The model tracks the effective survival rates on a stage class basis, as well as the sum and net dispersal distances. These values can be monitored and used as an aid in the selection of the input parameter values. Model parameterization

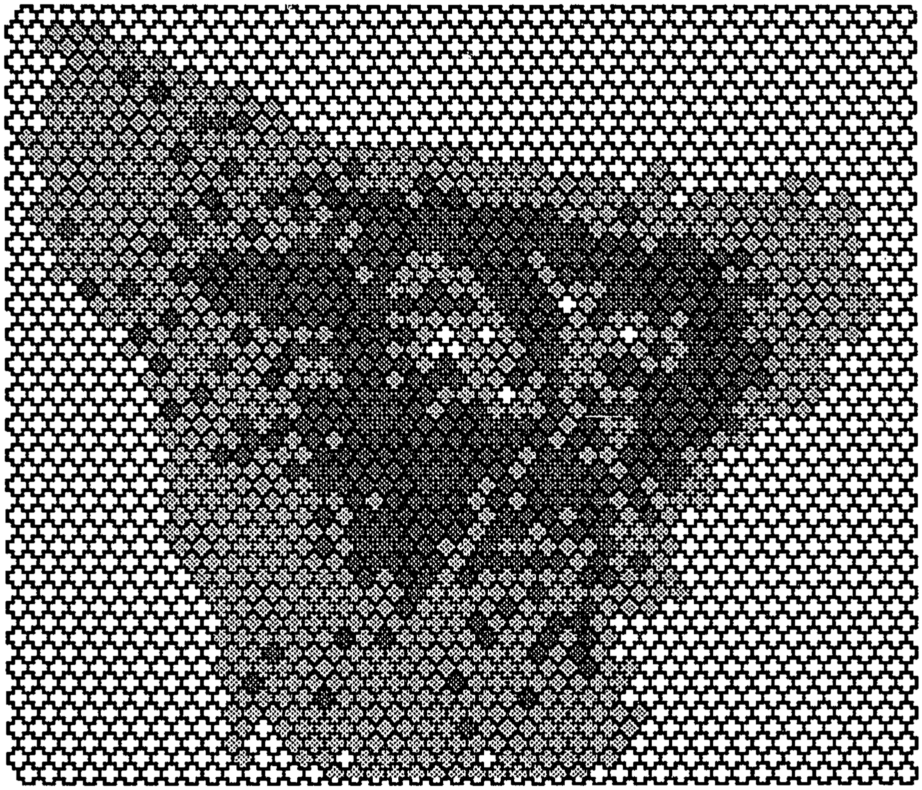


Figure 15. The territory map for the Olympic Peninsula used in the model simulations. Territories are not displayed hexagons in the figure due to a feature of the model's image compression facility. Territories in the computer's internal map remain hexagonal in shape. Suitable sites are dark grey in the figure. Sites possessing some habitat, but too little to function as a territory, are shown in light gray, and sites devoid of habitat are white.

was accomplished using the territory map derived from the Olympic Peninsula habitat coverage (figure 15), and by running the model for 100 replicate runs of 50 years.

The parameters specifying the minimum and maximum survival rates were adjusted so that the actual effective survival rates best matched those published in Burnham et al. (1994) for the Olympic Peninsula. Following Holthausen (1994) the maximum survival rate was set to 0.95 in all cases, and the minimum survival rate was then adjusted to obtain the desired realized mean survival. The target value for adult survival was taken from Burnham et al. (1994) to be 0.862. Subadult survival is thought to be lower than adult survival, but Burnham et al. (1994) did not detect a significant difference between the two. For this reason, the minimum and maximum subadult survival rates in the model were always set to those used for adult owls. Model output showed realized subadult survival to be lower than adult survival, but this discrepancy was minimized when a linear relationship was assumed between site quality and survival. For this reason the logistic fit to quality was not used in the model runs reported here. Data from Burnham et al. (1994) give a value for juvenile survival on the Olympic Peninsula between 0.245 and 0.582 depending on how juvenile emigration records are used in the computation. I found that if the effective adult survival was fixed at 0.862, this range of juvenile survivals was large enough to produce both stable and quickly declining owl populations. I developed a refined estimate for juvenile survival by running the model using a time series of changing GIS images representing habitat loss on the Olympic Peninsula between the period from 1921 to the present (to be discussed in full below). While running the model with these changing landscapes, I adjusted the juvenile survival rate to obtain a projection of today's population size that agreed with current estimates (Holthausen et al. 1994). No trend was observed in the effective survival rates for 100 replicate 50-year simulations that made use only of today's landscape (figure 16).

The mean number of movement steps was adjusted until the model reported a mean effective net dispersal distance as close as possible to 24.21 km, a value observed by Eric Forsman for dispersing spotted owls on the Olympic Peninsula (E. Forsman, unpublished data). Forsman's data also specifies minimum and maximum observed net dispersal distances of 8.67 and 58.24 km, respectively. Using δ to represent the value assigned to the mean number of dispersal steps, the model parameters specifying the minimum and maximum number of movement steps were thus set to (8.67/24.21)δ and (58.24/24.21)δ respectively. The movement linearity and bias to site quality were adjusted, along with the other movement parameters, to obtain the correct mean effective dispersal distance while at the same time obtaining a ratio of sum to net dispersal distances that fell within the range observed in the field (discussed below). Figure 17 displays the sum and net dispersal data resulting from this selection of parameters. The data in figure 17 appears stationary in time, and the growing variance results from the increased difficulty experienced by female floater owls attempting to locate territorial males in a steadily declining population. The variance in the male search paths decreased with time, but this is not evident in the figure.

E. Forsman (personal communication) suggests the ratio of sum to net dispersal distance for the Olympic Peninsula population could be as high as four. Data cited in Thomas et al. (1990) give an estimate of sum to net dispersal distance of 1.9. The final values chosen for movement linearity and bias to quality produce a ratio of sum to net dispersal distance of 2.4. This parameterization required selection of a relatively small bias to quality, and a large movement linearity, which together also generate the infrequent long distance dispersal events that have been documented in field studies (S. Horton, personal communication). Sex ratio was set to 50:50, and the maximum brood size and brood size probabilities were estimated from data in Forsman et al. (1984). The maximum brood size was set to 3, and the probabilities of a pair producing broods of sizes 1, 2, or 3 were set to 42%, 50%, and 8%, respectively. Because, in the model, reproductive output is not tied to habitat quality, the parameters specifying adult and subadult fecundity were set directly to the estimates from Burnham et al. (1994) for the Olympic Peninsula. Juvenile fecundity was rounded down to zero in all of the simulation runs. The model uses these fecundity values to compute the number of fledglings produced by paired females of each age class in aggregate, and then apportions these out randomly.

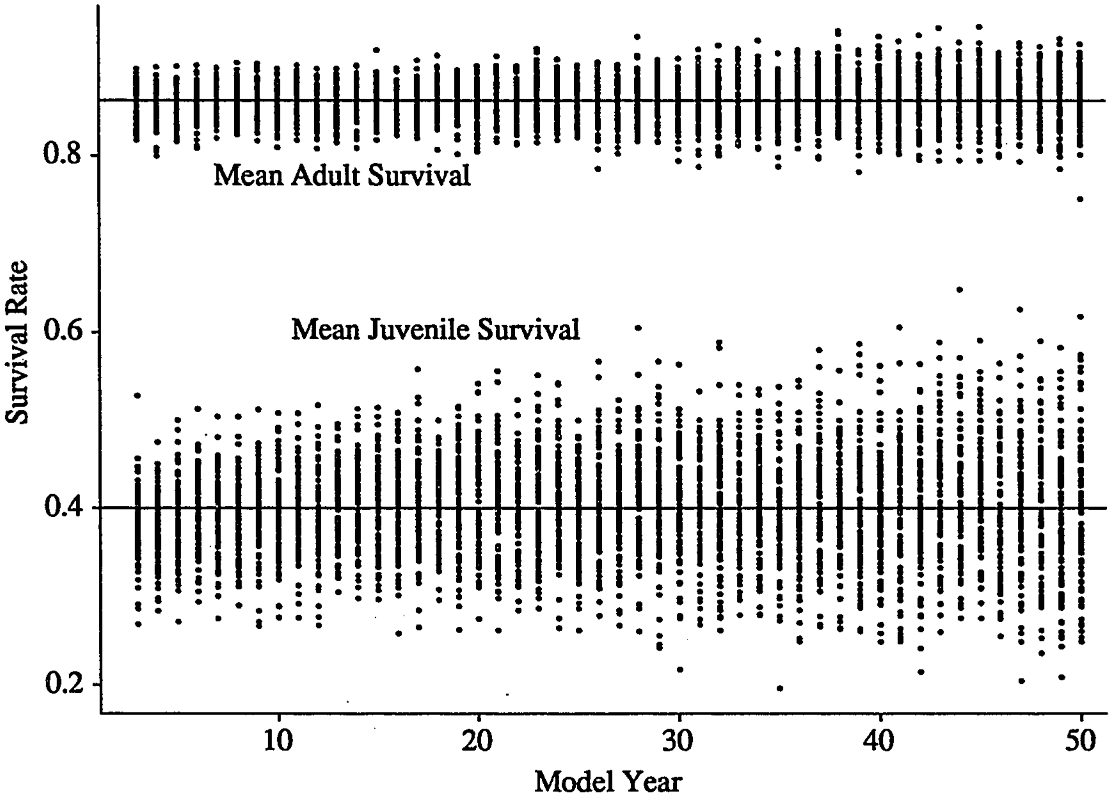


Figure 16. The mean adult and juvenile survival rates observed during the model calibration runs. These results are from 100 replicate runs of 50 years each. Each point in the figure represents a mean observed survival rate for a single year. Subadult survival is not shown in the figure because it overlaps significantly with the adult survival data. The observed survival rates stayed roughly constant with time, though the simulated owl population was experiencing a steep decline. The horizontal lines in the figure indicate the ensemble means, which are 0.862 for the adults, and 0.400 for the juvenile owls.

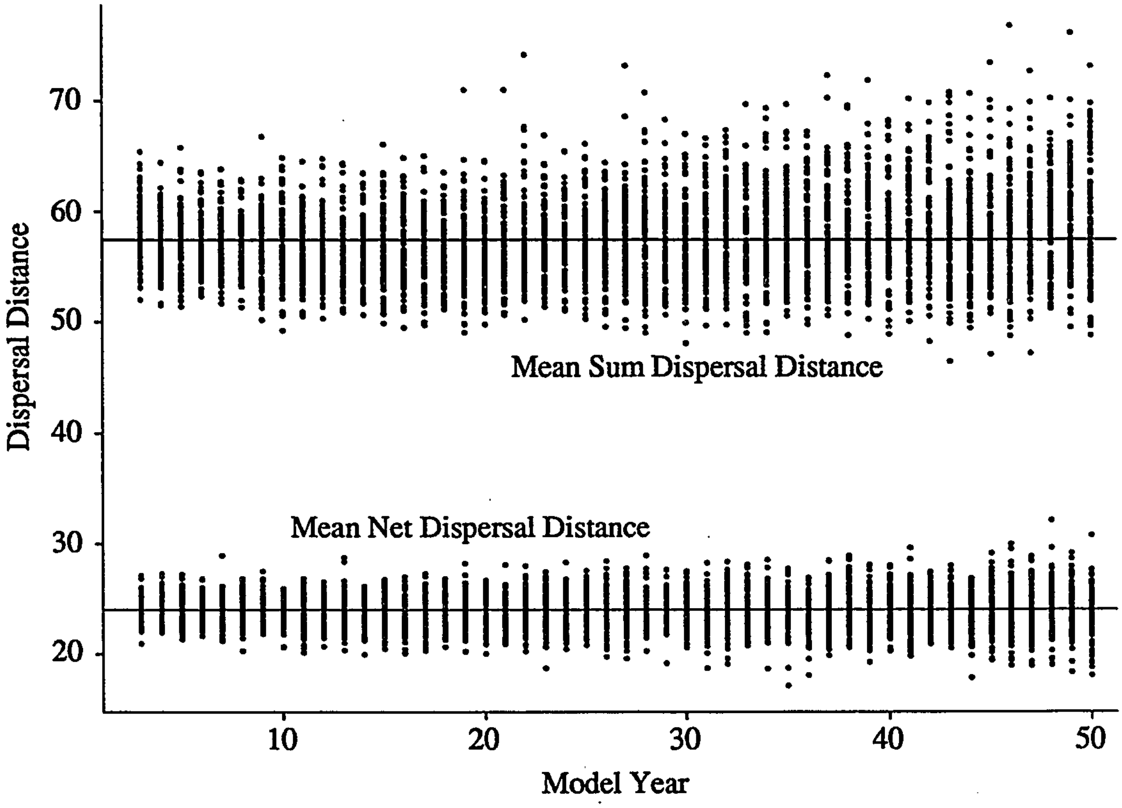


Figure 17. The mean dispersal distances observed during the model calibration runs. These results are from 100 replicate runs of 50 years each. Each point in the figure represents a mean observed dispersal distance for a single year. The upper collection of points shows the total distance traveled, while the lower collection of points shows the straight-line distance travelled. The horizontal lines in the figure indicate the ensemble means, which are 57.42 kilometers for sum distance, and 24.03 kilometers in the case of net distance.

Figure 18 shows the population trajectory that results from this combination of model parameters. It is clear from the figure that the initial number of owl pairs had a negligible effect on the projected 1994 population size. A second value of juvenile survival was also selected in order to produce a stable population of owls at the current estimated population size. This new value of juvenile survival is higher than the value described above, and the minimum value of adult survival had to be raised slightly as well to hold the realized value at 0.862. All other model parameters were held constant. The actual parameter values used in the simulation model are presented in table 3 (the survival rates in parentheses are the values used to produce a stable owl population).

Results

Sampling Efficiency

The projected number of spotted owls pairs was tracked by ownership for a 100 year period using the habitat map representing today's conditions. This analysis was conducted using the parameters described above, including both the value of juvenile survival obtained from the historic analysis (see methods) and the larger value that produced a stable owl population. Initially, these results were displayed as the total number of owl pairs present per year on the Olympic Park and wilderness areas, Olympic National Forest, and state, private and tribal lands (figures 19 and 20). All lands within the study area fall into one of these ownership classes, and thus the curves representing specific ownerships in figures 19 and 20 must sum to the curves depicting the entire peninsula's population. These population trajectories are smooth because they represent averages taken over 100 replicate model runs, and no trend is visible in the curves of figure 20 because these data were generated using a value of juvenile survival designed to produce a stable population. However, from figure 19 it is apparent that the declining owl population is shrinking within each ownership, and that the rate of decline of the entire population is

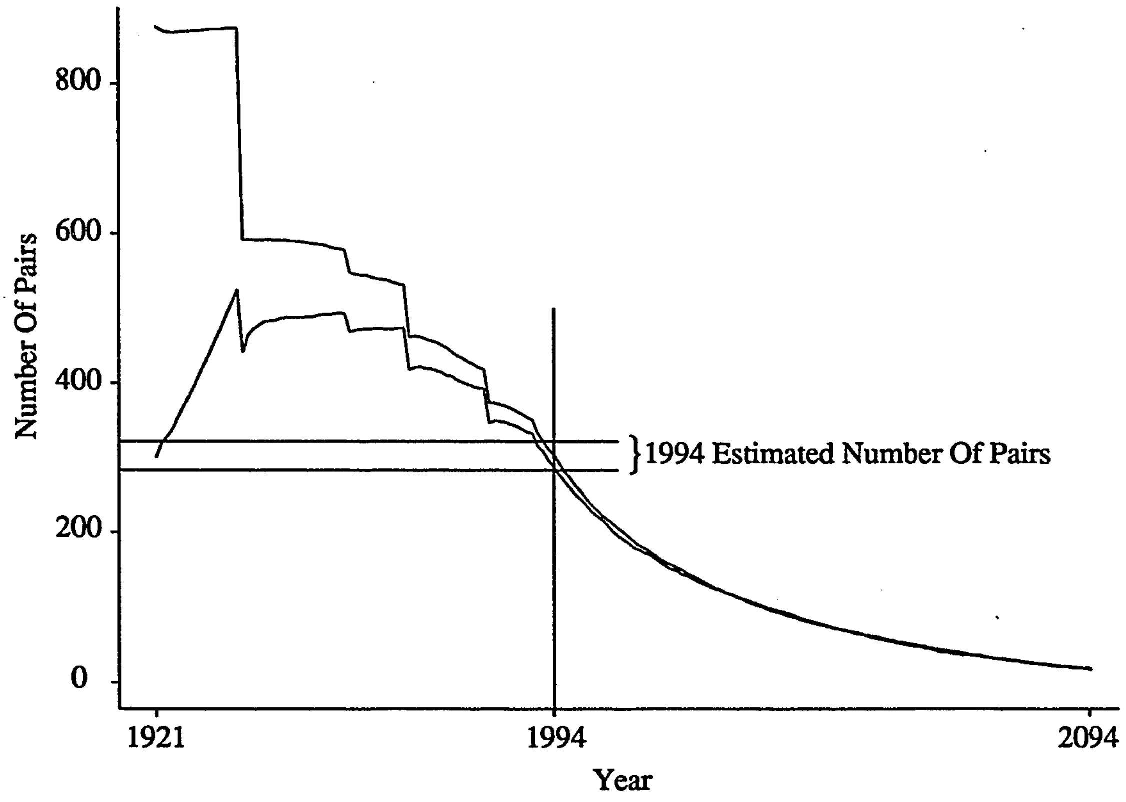


Figure 18. Numbers of spotted owl pairs as projected by the simulation model using a time series of Olympic Peninsula habitat data and the parameter set displayed in table 3. Different habitat maps were used in years 1921, 1936, 1956, 1967, 1982, 1991, and 1994. The curves in the figure are means from 100 replicate simulations. Juvenile survival was adjusted until the model's estimate of the number of spotted owl pairs in 1994 agreed with available data. The two curves in the figure demonstrate the model's insensitivity to the number of owl pairs placed into the landscape in 1921. A second value of juvenile survival was also selected that produced a stable population at approximately 300 pairs of owls (see methods), but the results from these simulations are not shown in the figure.

larger than the rate that would be estimated from any sub-sample of the population. This conclusion follows directly from the observation that each sample of the population displayed in figure 19 is in a monotonic decline. Throughout its range, the majority of sampling of spotted owls to date has been on lands owned by the U.S. Forest Service. Yet,

Table 3: Values Assigned To The Simulation Model Parameters

Parameter	Value
Minimum number of movement steps	11
Mean number of movement steps	31
Maximum number of movement steps	75
Movement linearity	90
Bias to site quality	10
Sex ratio	50:50
Maximum brood size	3
Probability of brood size of 1	0.42
Probability of brood size of 2	0.50
Probability of brood size of 3	0.08
Site fidelity minimum and maximum	50 - 100
Minimum & maximum juvenile survival	0.000 - 0.950 (0.180 - 0.950)
Minimum & maximum subadult survival	0.740 - 0.950 (0.760 - 0.950)
Minimum & maximum adult survival	0.740 - 0.950 (0.760 - 0.950)
Juvenile fecundity (males + females) / female	0.000
Subadult fecundity (males + females) / female	0.412
Adult fecundity (males + females) / female	0.760
Fit to site quality	Linear
Initialization method	Weighted
Boundary condition	Reflecting

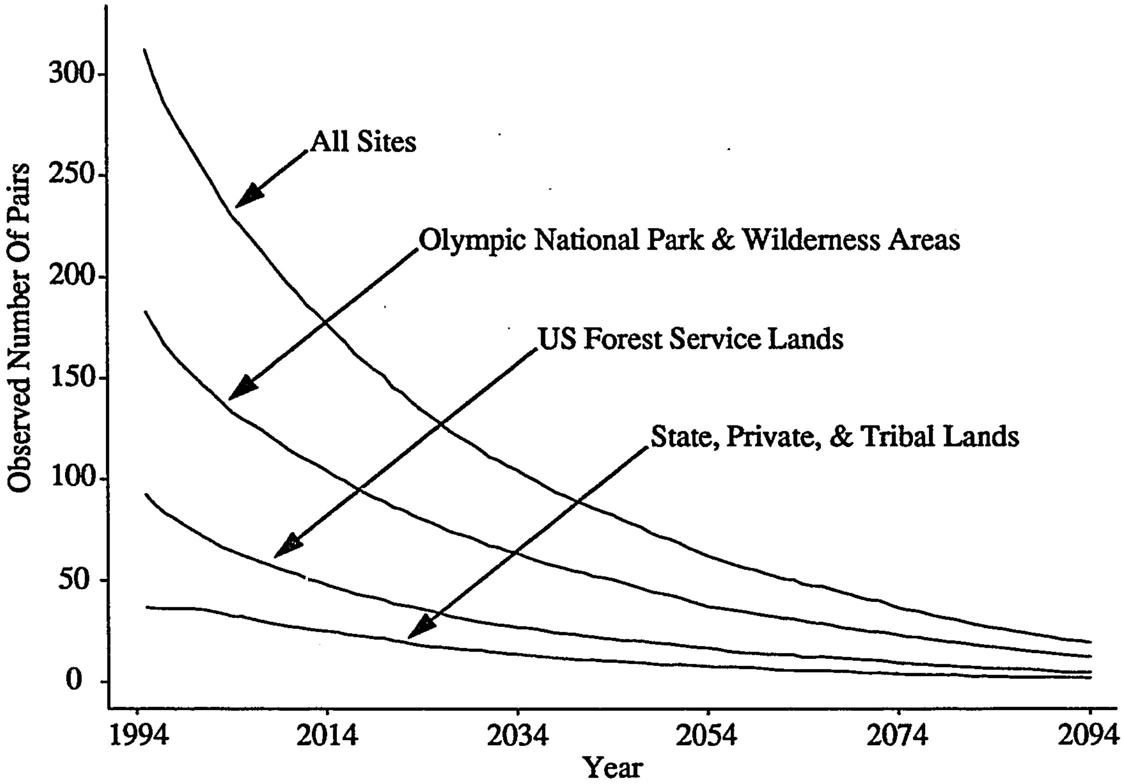


Figure 19. Results from sampling for the number of owl pairs peninsula-wide, and within three different ownership classes. These model runs were conducted using today's habitat data and the low estimate of juvenile survival (see methods), and represent means taken from 100 replicate simulations. None of the samples by ownership provide an acceptable approximation to the region-wide population size or trajectory.

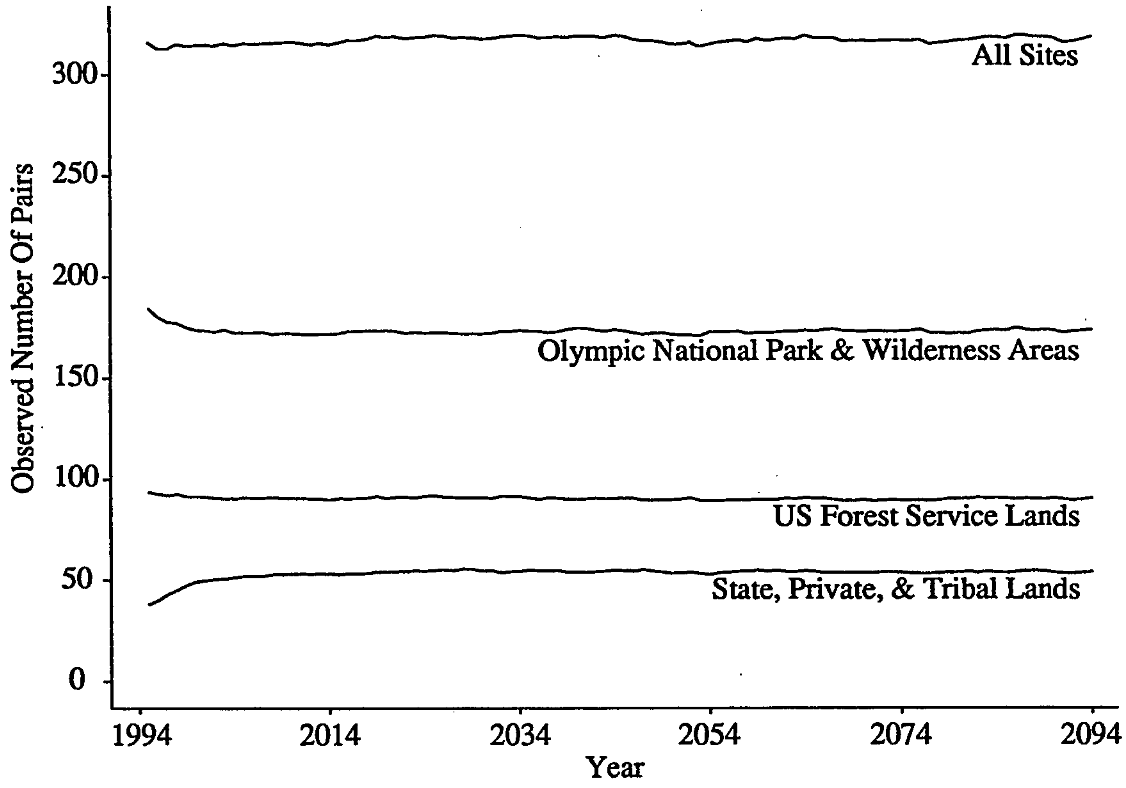


Figure 20. Results from sampling for the number of owl pairs peninsula-wide, and in three different ownership classes. These model runs were conducted using today's habitat data and the high estimate of juvenile survival (see methods), and represent means taken from 100 replicate simulations. None of the samples by ownership provide an acceptable approximation to the region-wide population size.

it is clear from figure 19 that the rate of decline of spotted owls on the Olympic Peninsula might be greatly underestimated if based on numbers of pairs observed on the Forest Service's lands alone.

The data presented in figures 19 and 20 were re-analyzed to display the fraction of suitable sites occupied by pairs, rather than the overall number of owl pairs. This simply entailed dividing the values of each curve in the figure by the total number of suitable sites in its respective ownership class. This changes the shapes of the resultant curves because the various ownership classes contain different numbers of suitable sites. Figures 21 and 22 display these rescaled population trajectories. These figures indicate that the rate of occupancy of suitable sites may function as a sensitive indicator of the true rate of occupancy. This was not found to be the case when the overall number of pairs of owls was the target of the investigation. The Forest Service's lands best follow the overall trend in occupancy rates, and this can be shown to be true because they better represent the peninsula-wide distribution of territory qualities than do the two other ownership classes. Figure 23 compares the distributions of territory qualities among the three ownerships, as well as for the entire Olympic Peninsula. It is apparent from this figure that the Olympic National Park and wilderness areas consist predominantly high quality territories, and that the state, private, and tribal lands hold a disproportionate number of poor quality territories. The distribution of territory qualities within Olympic National Forest is similar to that of the entire peninsula.

I used the procedure described above to examine the potential for even smaller collections of sites to predict region-wide population trends. I conducted this analysis using randomly selected territories, a collection of the best territories, and a collection of the worst territories. In each case, 50 sites were selected, and the analysis was conducted only for the declining owl population. The results of this comparison (figure 24) are the same as were those of figure 21. I repeated the analysis for ten collections of ten randomly selected sites (figure 25) and found that even a sample this small appears capable of picking out the trend in the region-wide population. These results suggest that a small sub-sample drawn

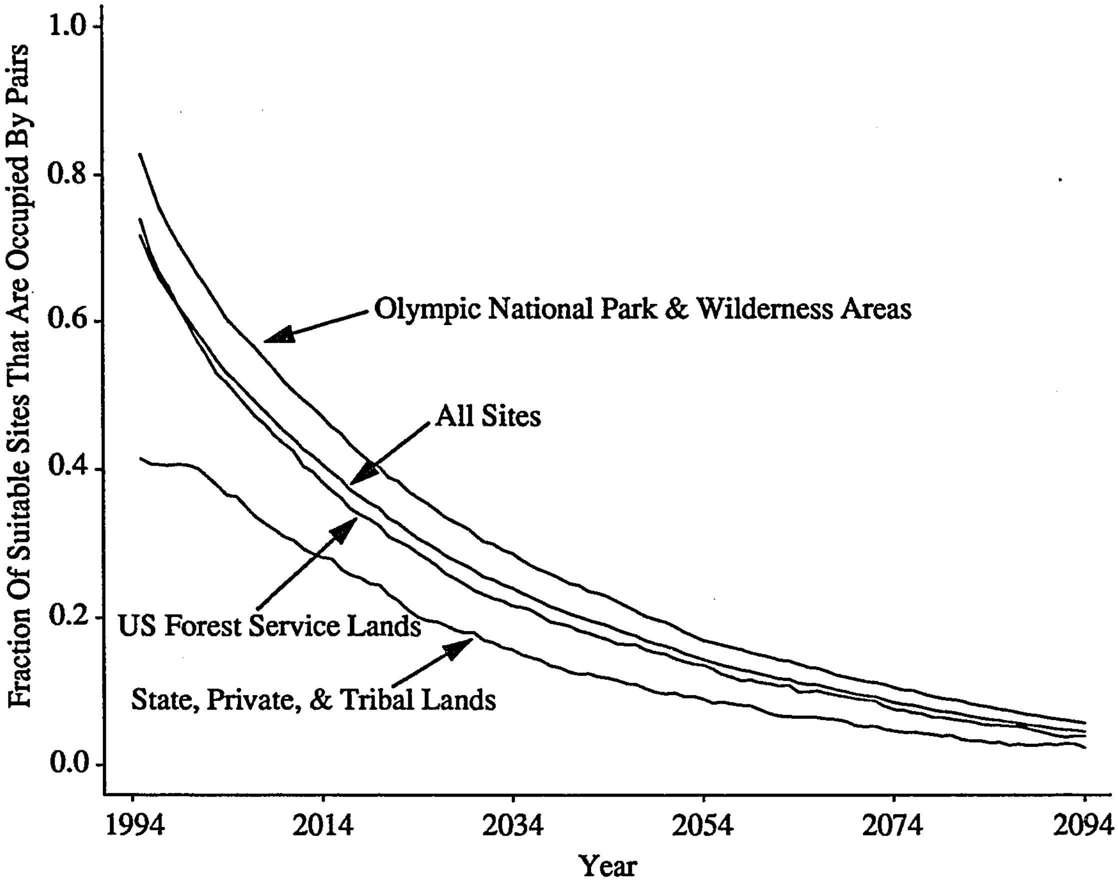


Figure 21. Results from sampling for the occupancy rate of owl pairs peninsula-wide, and in three different ownership classes. These model runs were conducted using today's habitat data and the low estimate of juvenile survival (see methods), and represent means taken from 100 replicate simulations. The data from the Olympic National Forest produce a good estimate of the peninsula-wide trends in occupancy rate.

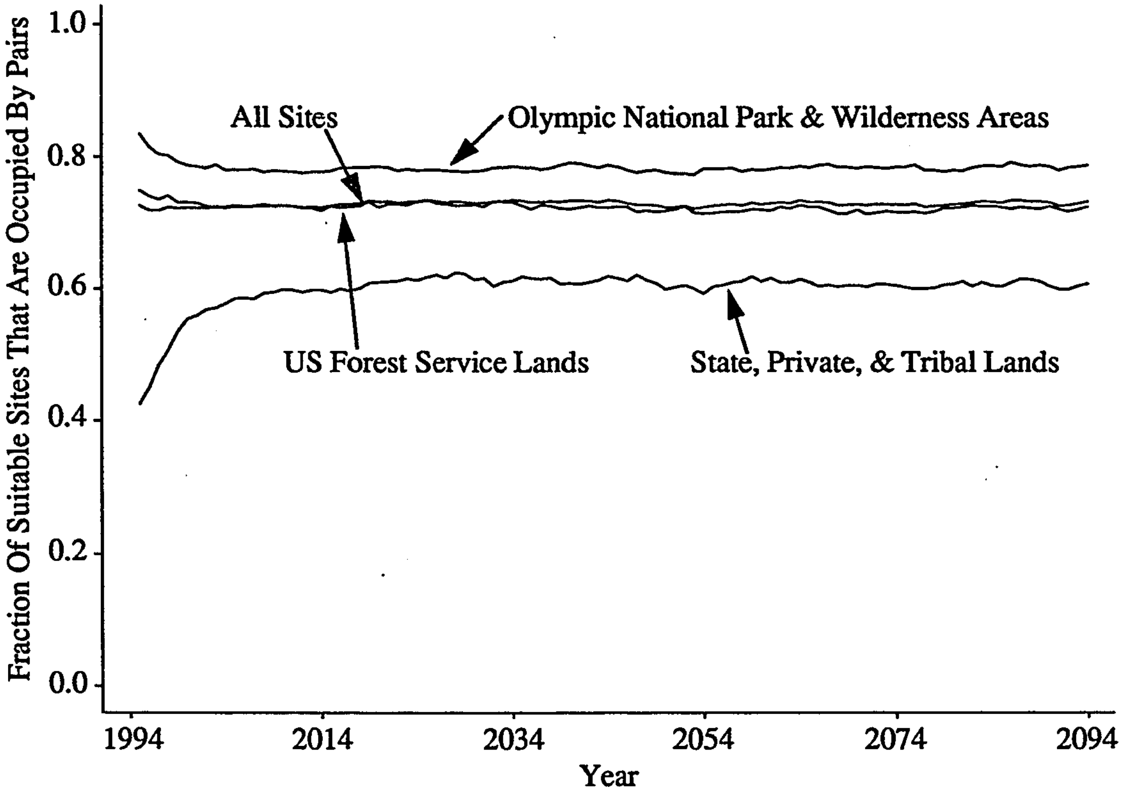


Figure 22. Results from sampling for the occupancy rate of owl pairs peninsula-wide, and in three different ownership classes. These model runs were conducted using today's habitat data and the high estimate of juvenile survival (see methods), and represent means taken from 100 replicate simulations. The data from the Olympic National Forest produce a good estimate of the peninsula-wide occupancy rate.

from the Olympic National Forest may be adequate to identify the peninsula-wide population trend if these sites are well distributed with respect to quality.

A second conclusion of this analysis is that the present estimates of trends in spotted owl populations throughout the species' range are likely to be quite good. Throughout the Pacific Northwest, much of the data used to estimate trends in spotted owl populations has been collected on U.S. Forest Service lands (Holthausen et al 1994, Burnham et al. 1994), and these lands typically represent a spectrum of habitat qualities, just as does the Olympic National Forest (Holthausen et al. 1994, Carey et al. 1992). (The majority of the Pacific Northwest's remaining high quality spotted owl habitat is presently on Forest Service lands, but the agency also owns a great deal of moderate and marginal quality lands throughout the region.) A simple check on the above analysis can be made as follows: Holthausen et al. (1994) provide low and high estimates of the number of pairs of spotted owls on the Olympic Peninsula of 282, and 321 pairs. If the occupancy rate of spotted owls on Forest Service lands does truly approximate that of the entire region, then it should be correct to write

$$\frac{\text{Number ONF Pairs}}{\text{Suitable ONF Sites}} \approx \frac{\text{Number OLP Pairs}}{\text{Suitable OLP Sites}}$$

where ONF and OLP refer respectively to the Olympic National Forest, and the Olympic Peninsula. Assuming the mean territory sizes are the same, and solving for the peninsula-wide population gives:

$$\text{Number OLP Pairs} \approx \frac{(\text{ONF Pairs})\ (\text{OLP NRF Habitat})}{\text{ONF NRF Habitat}}$$

where NRF refers to nesting, roosting, and foraging habitat. Holthausen et al. (1994) provide estimates of NRF habitat by ownership across the Olympic Peninsula including values for the Olympic National Park and Olympic National Forest of 278,000 and 97,600

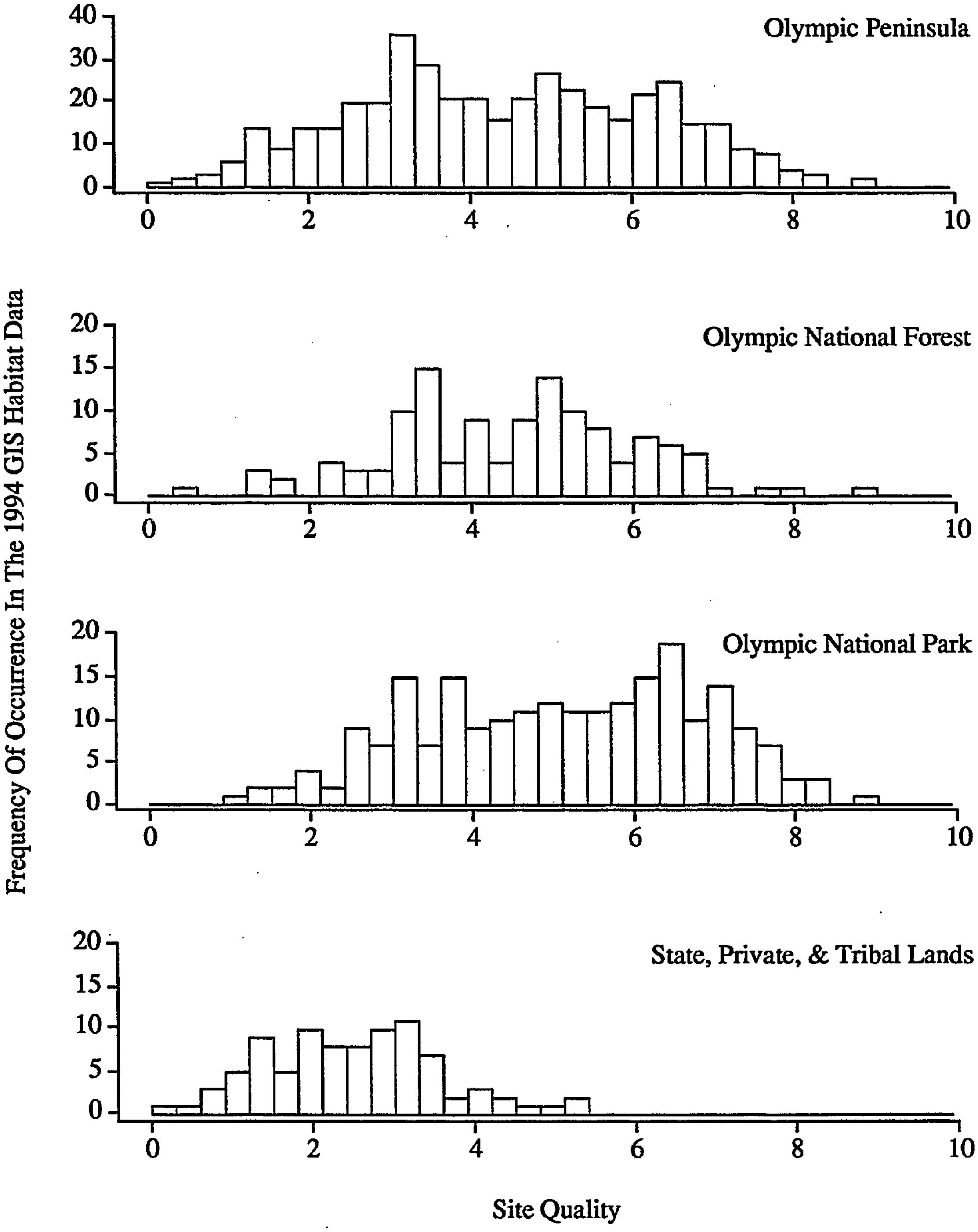


Figure 23. Histograms showing the frequency of suitable site qualities by ownership for the 1994 Olympic Peninsula GIS habitat data. Sites that did not qualify as territories are not included in the histograms.

hectares respectively. When entered into the above equation, these values yield an estimate of the peninsula-wide population of:

$$\text{Number OLP Pairs} \approx 2.85 \cdot \text{Number ONF Pairs}\,.$$

Holthausen et al. (1994) set low and high estimates of the number of pairs of spotted owls on Olympic National Forest lands at 97, and 117 pairs. Using each of these values in the above relation gives low and high estimates of the region-wide population of 276 and 333 pairs, in close agreement with the cited values of 282 and 321 pairs. The Olympic National Forest comprises 21% of the land area of the Olympic Peninsula and is 70% of the size of the Olympic National Park (Holthausen et al. 1994).

Estimating the actual number of suitable sites in a real landscape can prove a difficult task for ownerships as large as the Olympic National Forest. But the results of figure 25 demonstrate that a good estimate of the region-wide population might also be obtained using a sub-sample from a known number of sites that reflect the overall range of territory qualities available. Tracking the occupancy of a limited number of territories is a feasible task, and a simple check on the adequacy of the sample could be conducted as was done above to see if it produces an estimate of the region-wide population that is within acceptable bounds.

Spotted owls on the Olympic Experimental State Forest

The Washington State Department of Natural Resources owns more than 100,000 hectares of forest resource land on the western Olympic Peninsula, collectively referred to as the Olympic Experimental State Forest (OESF). One of the legislated management goals of the OESF is that it meet the recovery objectives specified in the federal recovery plan for the northern spotted owl for the western Olympic Peninsula. The Washington DNR is

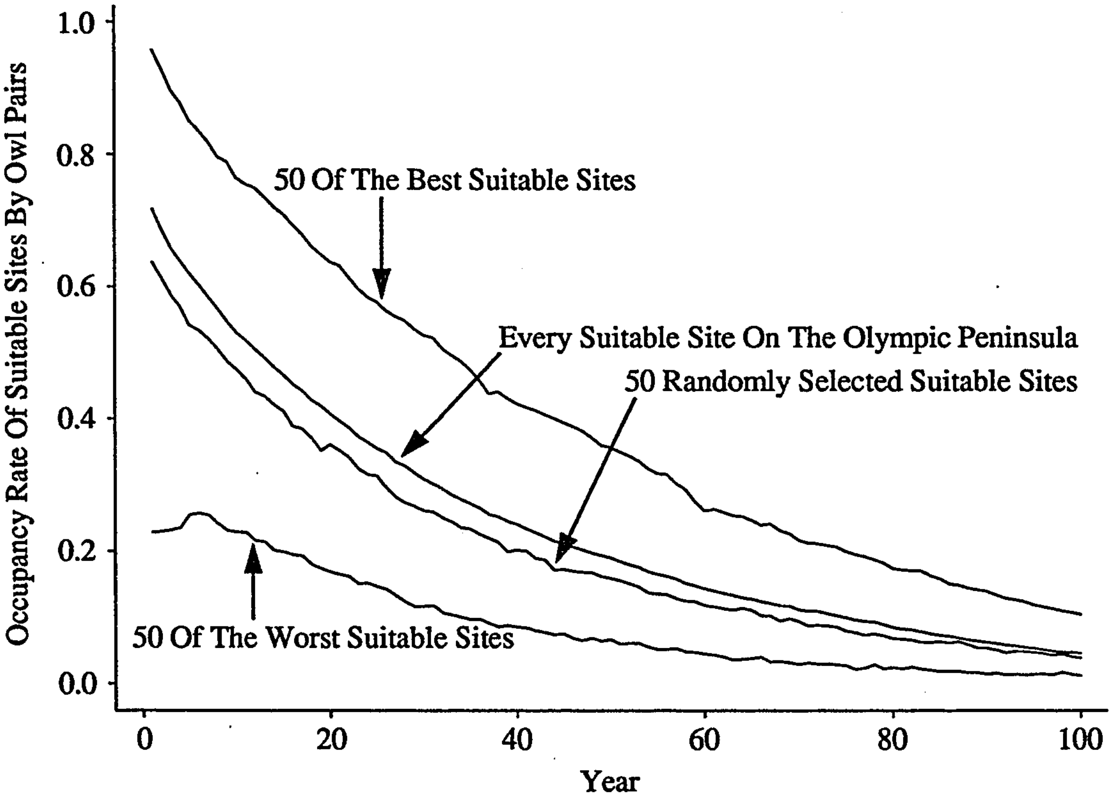


Figure 24. The projected peninsula-wide occupancy rate of suitable sites by owl pairs, and the occupancy rates observed in three different samples of 50 sites. These curves represent mean values taken from 100 replicate model runs conducted using today's landscape.

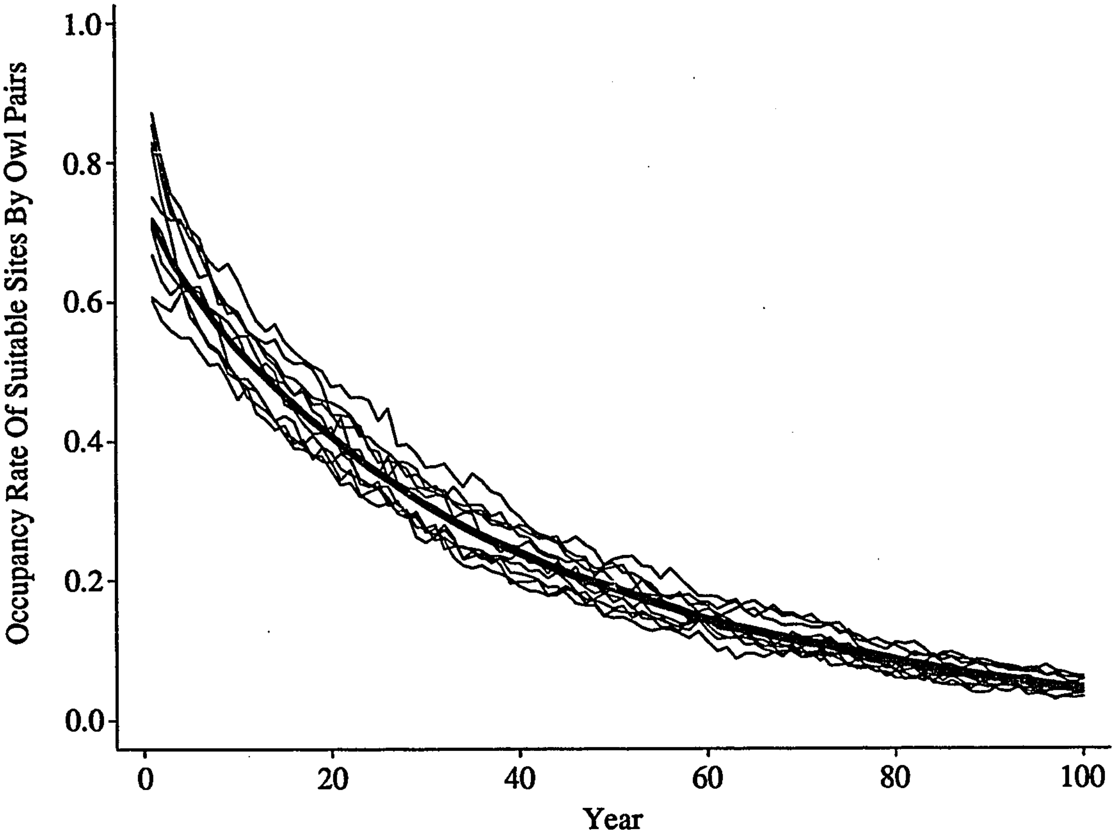


Figure 25. The projected peninsula-wide occupancy rate of suitable sites by owl pairs, and the occupancy rates observed in ten different samples of ten sites. The thick line in the center of the collection of trajectories represents the peninsula-wide population. These curves represent mean values taken from 100 replicate model runs conducted using today's landscape.

considering three different management scenarios for the OESF that all reflect this legislative mandate to some degree. These three strategies include management as usual, initial loss of some existing owl habitat followed by complete protection, and immediate and complete protection of all existing owl habitat. Of these three plans, the perpetuation of today's management style constitutes the least conservative strategy in regards to spotted owl habitat because it calls only for protection of existing habitat surrounding currently occupied owl nests. The other two proposed plans include some level of retention of current owl habitat plus silvicultural prescriptions intended to promote the regeneration of at least 40% nesting, roosting, and foraging habitat within each of the OESF's 11 planning units. These management strategies differ in that one allows a short term loss of existing owl habitat whereas the other does not.

Personnel from the Washington Department of Natural Resources used projections of habitat change under the three scenarios described above to develop territory maps representing each management strategy at 25, 50, and 75 years into the future. In developing these territory maps, it was assumed that on average the park, wilderness, state, and tribal lands would not change with time. The amount and distribution of suitable spotted owl habitat within the Olympic National Forest was made to increase through time in the manner specified by the Federal Forest Plan (USDA et al. 1993, USDA/USDI 1994), and all three of the OESF management strategies examined here incorporate this change. Model runs that used only the territory map representing today's distribution of habitat were occasionally employed as controls.

The three different OESF land management strategies were compared using the model parameterization that generated a declining owl population with the Olympic Peninsula GIS data (see methods). Figure 26 displays the projected numbers of spotted owl pairs under each of these management strategies, as well as the trajectory that results when the landscape is fixed in today's conditions. The analysis of figure 26 was conducted by starting with the 1921 landscape and progressing through a series of landscape changes that eventually arrived at the territory map representing current conditions (see methods for details). The curves in figure 26 are smooth because they represent the values from 100 replicate simulations. All three of the trajectories representing some improvement in habitat through time show the owl population entering a period of recovery. In contrast, the trajectory generated with the static landscape declines towards extinction. None of the 300 model runs conducted in this examination of the three potential OESF management strategies produced an owl population that went extinct.

To more closely examine the differences between the three land management scenarios, the above analysis was repeated starting with the today's landscape and running for 300 years into the future (the control, for which the landscape did not change, was left out of this analysis). As was done previously, 100 replicate simulations were conducted for each strategy, and the results are displayed in figure 27 in the form of averages. The landscape changes only at 25, 50, and 75 years into the future, but the simulations here are carried out for enough time to make apparent the ultimate differences in the owl's response to the management strategies. For the time period in which they overlap, the top and bottom curves in figure 27 are identical to those generated starting with the 1921 landscape (figure 26). The center curve in figure 27 lies slightly above its counterpart in figure 26. In all three cases, the model projects a minimum population that includes, on average, only 100 pairs of owls, but in each case the recovery appears to last. As was true in the analysis that began with the 1921 landscape, none of the 300 model runs used in this analysis produced an extinction.

Perhaps the most significant feature of figure 27 is that the spotted owl's projected response to landscape change is much larger than what would be predicted based solely on the development of additional suitable habitat. Once the territory map has stopped changing, the bottom curve in figure 27 represents a landscape containing 469 suitable sites, whereas the top curve represents 512 suitable sites. At 300 years in the future, the two curves are separated by 100 owl pairs, or 2.3 times that which would be expected based strictly on changes in available habitat alone. The owl's dramatic response reflects an increase in habitat connectivity that was sufficient to allow a larger fraction of the

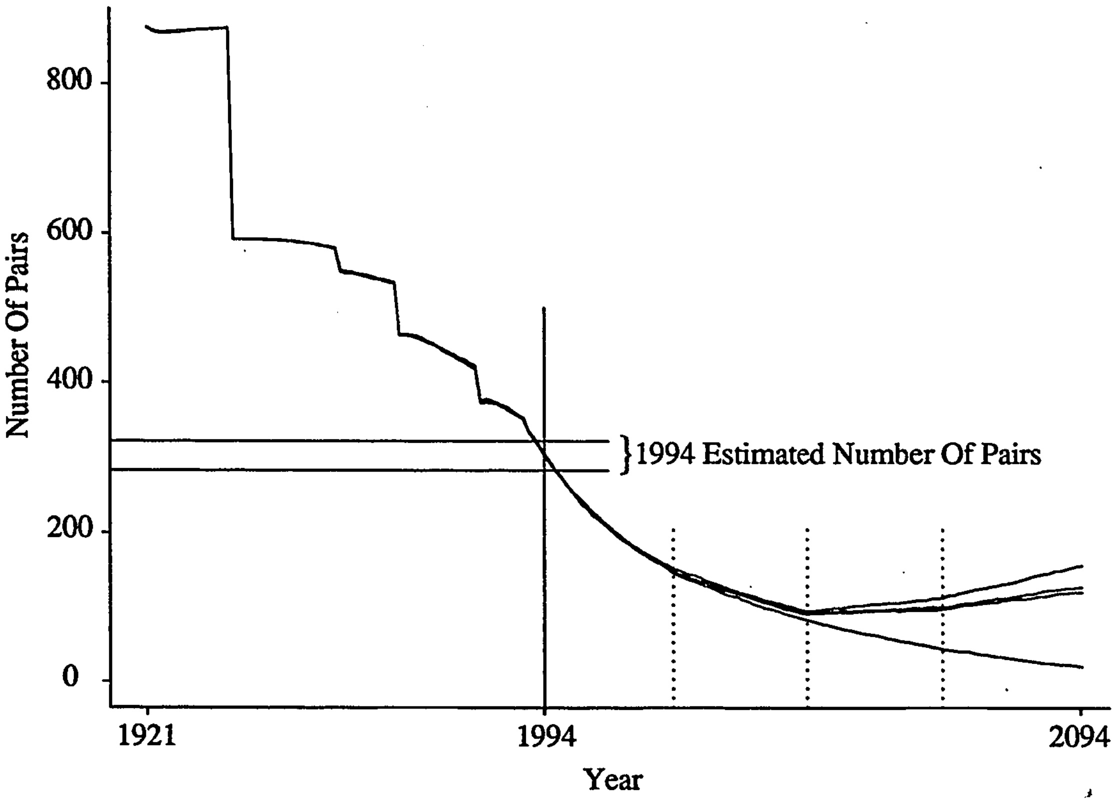


Figure 26. Changes in the projected numbers of owl pairs resulting from the three proposed Olympic Experimental State Forest management strategies. The population trajectories in the figure are each mean values taken from 100 replicate simulations. The habitat map changes at years 1921, 1936, 1956, 1967, 1982, 1991, and 1994. The three OESF management strategies are simulated through additional habitat modifications made at 25, 50, and 75 years into the future, as indicated by the dotted vertical lines in the figure. The lower curve is obtained by progressing through the same series of habitat maps up to 1994, and then holding the landscape in its current condition. The lower curve in this figure is identical to the upper curve of figure 18.

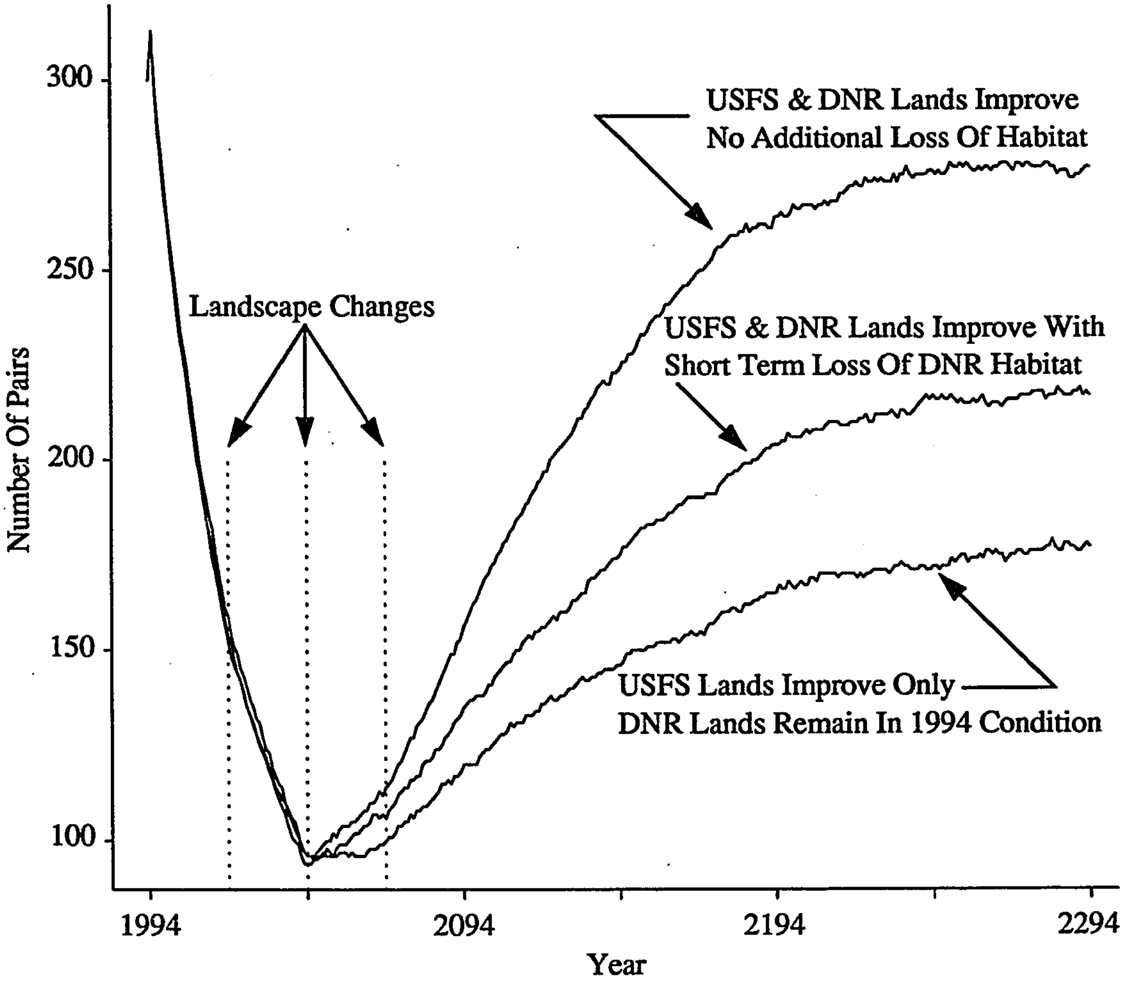


Figure 27. The projected long-term population-level responses to the three proposed Olympic Experimental State Forest management plans. The population trajectories in the figure each represent means taken from 100 replicate simulations. These model runs were started with the 1994 landscape, and the three management strategies are simulated through changes made to the territory map at 25, 50, and 75 years into the future.

suitable sites already present to become occupied.

The results of figures 26 and 27 suggest that even if the current rate of decline is as fast as is suggested by figure 18, the Olympic Peninsula spotted owl population may still not yet have crossed a threshold beyond which there is no chance of recovery. In addition, it appears that Washington State has the potential to contribute to the Olympic Peninsula spotted owl population to a degree that far exceeds what might be expected based on today's distribution of habitat.

Discussion

Field studies have generated strong evidence that the northern spotted owl is declining throughout its range (Burnham 1994). These studies have typically produced estimates of survival, fecundity, and dispersal success rates too low to support a stable population (Thomas et al. 1990, Burnham 1994). But as good as these data are, it has still proven quite difficult to estimate what the size and distribution of the owl population will be in 50 or 100 years. And predictions of population trends through time become harder to make when a only subset of the population is examined, because significantly less data is then available, and because local trends do not necessarily follow regional ones. Isolation and unusual habitat patterns make this is the case for the spotted owl on the Olympic Peninsula of Washington State. In time, enough additional data may be collected that local trends in spotted owl numbers become clear, but decisions that limit options for the future are being based on the today's estimates of the magnitude of the threat to the species. Though data for the spotted owl on the Olympic Peninsula are remarkable good, the fact that they have come largely from studies taking place on U.S. Forest Service lands has been a source of considerable scepticism. To date, no consensus even exists on whether the Olympic Peninsula owl population is in a steady state, or in a decline (S. Horton, personal communication).

Results from this study indicate that exhaustive sampling is probably not necessary to obtain a reliable estimate of the size or rate of change of the Olympic Peninsula population of spotted owls. A sub-sample that accurately reflects the spectrum of territory qualities may provide an adequate indication of trends in the larger population. The lands of the Olympic National Forest appear to qualify as such a sample, and an estimate of the peninsula-wide owl population based strictly on the number of owls observed in the National Forest confirms this assertion. This study also demonstrates that a sample of the population much smaller than the size of the Olympic National Forest may also provide a good estimate of trends in the larger population. These results suggest that, if the resources for monitoring are limited, it may be more worthwhile to carefully conduct limited sampling, and to place additional effort into estimating territory qualities region-wide, rather than attempt to expand existing sampling studies into non-federal ownerships, and parks, as is currently being considered (M. Raphael, personal communication).

Several conclusions may be drawn from the model runs that incorporate habitat change through time. The use of a sequence of changing landscapes beginning in 1921 and progressing to the present made it possible to select a value for juvenile survival from the range suggested by currently available field data. An important consequence of conducting the model parameterization and analysis in this manner is that the results appear to support the notion that the spotted owl is currently in a state of decline on the Olympic Peninsula. This methodology could both be repeated in other situations where data limitations make model parameterization difficult, and could easily be improved through the development of better historic GIS habitat data.

The parameterized simulation model was used with three projections of future landscape change that mimic potential management strategies of the Washington DNR. The model population of spotted owls reversed its decline under each of these three management scenarios, and this analysis indicates that the Olympic Peninsula owl population may not yet have crossed a threshold in size beyond which recovery is impossible. In addition, the difference in the projected number of pairs of spotted owls (at 300 years in the future) between the least and most conservative DNR strategies was more than double the change in suitable sites. This compensatory response to an increase in habitat connectivity indicates that the Washington State forest lands may play a larger role in the fate of the spotted owl on the Olympic Peninsula than would be predicted based strictly on projections of habitat regeneration.

Literature Cited

Adler, F. R., and B. Nuernberger. 1994. Persistence in patchy irregular landscapes. Theoretical Population Biology **45**:41-75.

Bart, J. In Press. Amount of suitable habitat and viability of northern spotted owls. Conservation Biology.

Bart, J., and E. D. Forsman. 1992. Dependence of northern spotted owls *Strix occidentalis caurina* on old-growth forests in the western USA. Biological Conservation **62**: 95-100.

Boyce, M. S. 1987. A review of the U.S. Forest Service's viability analysis for the Spotted Owl. Final report to the National Council of the Paper Industry for Air and Stream Improvement.

Buechner, M. 1987. Conservation in insular parks: simulation models of factors affecting the movement of animals across park boundaries. Biological Conservation **41**:57-76.

Buechner, M. 1989. Are small-scale landscape features important factors for field studies of small mammal dispersal sinks? Landscape Ecology **2**:191-199.

Burel, F. 1989. Landscape structure effects on carabid beetles spatial patterns in western France. Landscape Ecology 2:215-226.

Burnham, K. P., D. R. Anderson, and G. C. White. 1994. Estimation of vital rates of the northern spotted owl. Appendix J, pages 1-26 *in* USDA Forest Service, Final Supplemental Environmental Impact Statement. Portland, OR. Volume II.

Carey, A. B., S. P. Horton, and B. L. Biswell. 1992. Northern spotted owls: influence of prey base and landscape character. Ecological Monographs **6**:223-250.

Chen, J. 1991. Microclimatic and biological pattern at edges of Douglas-fir stands. Dissertation. University of Washington, Seattle, Washington, USA.

Chen, J., Franklin, J. F., and T. A. Spies. 1992. Vegetation responses to edge environments in old-growth Douglas-fir forests. Ecological Applications **2**:387-396.

Doak, D. 1989. Spotted Owls and old growth logging in the Pacific Northwest. Conservation Biology **3**:389-396.

Doak, D. F., P. C. Marino, and P. M. Kareiva. 1992. Spatial scale mediates the influence of habitat fragmentation on dispersal success: implications for conservation. Theoretical Population Biology **41**:315-336.

Doak, D. F. and L. S. Mills. 1994. A useful role for theory in conservation. Ecology **75**:615-626.

Duever L. C. and R. F. Noss. 1990. A computerized method of priority ranking for natural areas. Pages 22-33 *in* Ecosystem management: rare species and significant habitats. New York State Museum bulletin 471.

Dunning, J. B., D. J. Stewart, B. J. Danielson, B. R. Noon, T. Root, R. H. Lamberson, and E. E. Stevens. *In press*. Spatially-explicit population models: current forms and future uses.

Fahrig, L. and G. Merriam. 1985. Habitat patch connectivity and population survival. Ecology **66**: 1762-1768.

Fahrig, L., and J. Paloheimo. 1988*a*. Effect of spatial arrangement of habitat patches on local population size. Ecology **69**:468-475.

Fahrig, L. and J. Paloheimo. 1988*b*. Determinants of local population size in patchy habitats. Theoretical Population Biology **34**:194-213.

Forsman, E. D., E. C. Meslow, and H. M. Wight. 1984. Distribution and biology of the spotted owl in Oregon. Wildlife Monographs **87**:1-64.

Forman, R. T. T. and M. Godron. 1986. Landscape Ecology. John Wiley & Sons, New York, New York. USA.

Foster, J., and M. S. Gaines. 1991. The effects of a successional habitat mosaic on a small mammal community. Ecology **72**:1358-1373.

Franklin, J. F., and R. T. T. Forman. 1987. Creating landscape patterns by forest cutting: Ecological consequences and principles. Landscape Ecology **1**:5-18.

Gardner R. H., and R. V. O'Neill. 1991. Pattern, process and predictability: the use of neutral models for landscape analysis. Pages 289-307 *in* M. G. Turner, and R. H. Gardner, editors. Quantitative methods in landscape ecology. Springer-Verlag, New York, New York, USA.

Groom, M. J., and N. Schumaker. 1993. Evaluating landscape change: patterns of worldwide deforestation and local fragmentation. Pages 24-44 *in* P. M. Kareiva, J. G. Kingsolver, and R. B. Huey, editors. Biotic interactions and global change. Sinauer, Sunderland, Massachusetts, USA.

Gustafson, E. J., and G. R. Parker. 1992. Relationships between landcover proportion and indices of landscape spatial pattern. Landscape Ecology **7**:101-110.

Gutiérrez, R. J., A. B. Franklin, W. Laheye, V. J. Meretsky, and J. P. Ward. 1985. Juvenile spotted owl dispersal in northern California: preliminary results. Pages 60-65 *in* R. J. Gutiérrez and A. B. Carey editors. Ecology and management of the spotted owl in the Pacific Northwest. General Technical Report PNW-185. USDA Forest Service, Pacific Northwest Research Station. Portland, Oregon.

Hansen A. J., and Urban D. L. 1992. Avian response to landscape pattern: the role of species' life histories. Landscape Ecology **7**:163-180.

Harrison, S. 1989. Long-distance dispersal and colonization in the Bay checkerspot butterfly, *Euphydryas editha bayensis*. Ecology **70**:1236-1243.

Hastings, A., and C. L. Wolin. 1989. Within-patch dynamics in a metapopulation. Ecology **70**:1261-1266.

Henderson, J. A., D. H. Peter, R. D. Lesher, and D. C. Shaw. 1989. Forested plant associations of the Olympic National Forest. USDA Forest Service Pacific Northwest Region, R6-ECOL-TP 001-88.

Henein, K., and G. Merriam. 1990. The elements of connectivity where corridor quality is variable. Landscape Ecology **4**:157-170.

Holthausen, R. S., M. G. Raphael, K. S. McKelvey, E. D. Forsman, E. E. Starkey, and D. E. Seaman. 1994. The contribution of federal and nonfederal habitat to persistence of the northern spotted owl on the Olympic Peninsula, Washington. Report of the Reanalysis Team. 76 pages plus appendices.

Houston, B. R., T. W. Clark, and S. C. Minta. 1986. Habitat suitability index model for the black-footed ferret: a method to locate transplant sites. Great Basin Natural Memoirs **8**:99-114.

Johnson, A. R., B. T. Milne, and J. A. Wiens. 1992. Diffusion in fractal landscapes: simulations and experimental studies of tenebrionid beetle movements. Ecology **73**:1968-1983.

Kareiva, P. 1987. Habitat fragmentation and the stability of predator-prey interactions. Nature **326**:388-390.

Kareiva, P. 1990. Population dynamics in spatially complex environments: theory and data. Philosophical Transactions of the Royal Society of London. B **330**, 175-190.

Kareiva, P. and U. Wennegren. *In review.* Connecting landscape patterns to ecosystem and population processes.

Keller J. K. 1990. Using aerial photography to model species-habitat relationships: the importance of habitat size and shape. Pages 34-46 *in* Ecosystem management: rare species and significant habitats. New York State Museum bulletin 471.

Lamberson R. H., R. McKelvey, B. R. Noon, and C. Voss. 1992. A dynamic analysis of Northern Spotted Owl viability in a fragmented forest landscape. Conservation Biology **6**:505-512.

Lande, R. 1987. Extinction thresholds in demographic models of territorial populations. American Naturalist **130**:624-635.

Lande, R. 1988. Demographic models of the Northern Spotted Owl (*Strix occidentalis caurina*). Oecologia **75**:601-607.

Li, H. and J. F. Reynolds. 1993. A new contagion index to quantify spatial patterns of landscapes. Landscape Ecology **8**:155-162.

Marcot, B. G., and R. Holthausen. 1987. Analyzing population viability of the Spotted Owl in the Pacific Northwest. Transactions of the North American Wildlife and Natural Resources Conference **52**:333-347.

Marcot, B. G., R. S. Holthausen, J. Teply, and W. D. Carrier. 1991. Old-growth inventories: status, definitions, and visions for the future. Pages 47-60 *in* L.F. Ruggiero, K.B. Aubry, A.B. Carey, and M.H. Huff, editors. Wildlife and vegetation of unmanaged Douglas-fir forests. USDA, Forest Service PNW-GTR-285, Pacific Northwest Research Station, Portland, Oregon, USA.

McKelvey, K., B. R. Noon, and R. H. Lamberson. 1993. Conservation planning for species occupying fragmented landscapes: the case of the northern spotted owl. Pages 424-450 *in* P. M. Kareiva, J. G. Kingsolver, and R. B. Huey, editors. Biotic interactions and global change. Sinauer, Sunderland, Massachusetts, USA.

Miller R. I., S. N. Simon, and K. M. Howell. 1989. A methodology for analyzing rare species distribution patterns utilizing GIS technology: the rare birds of Tanzania. Landscape Ecology 2:173-189.

Milne, B. T. 1988. Measuring the fractal geometry of landscapes. Applied Mathematics and Computation 27:67-79.

Milne, B. 1991. Lessons from applying fractal models to landscape patterns. Pages 199-235 *in* M. G. Turner, and R. H. Gardner, editors. Quantitative methods in landscape ecology. Springer-Verlag, New York, New York, USA.

Noon, B. R., and C. M. Biles. 1990. Mathematical demography of spotted owls in the Pacific Northwest. Journal of Wildlife Management 54:18-27.

Noss, R. F. 1990. Indicators for monitoring biodiversity: a hierarchical approach. Conservation Biology 4:355-364.

Old-Growth Definition Task Group. 1986. Interim definitions for old-growth Douglas-fir and mixed conifer forests in the Pacific Northwest and California. PNW Research Station. Res. Note PNW-447. Portland, OR.

O'Neill, R. V., J. R. Krummel, R. H. Gardner, G. Sugihara, B. Jackson, D. L. DeAngelis, B. T. Milne, M. G. Turner, B. Zygmunt, S. W. Christensen, V. H. Dale, and R. L. Graham. 1988. Indices of landscape pattern. Landscape Ecology 1:153-162.

Patton, D. R. 1975. A diversity index for quantifying habitat "edge". Wildlife Society Bulletin 3: 171-173.

Plotnick, R. E., R. H. Gardner, and R. V. O'Neill. 1993. Lacunarity indices as measures of landscape texture. Landscape Ecology **8**:201-211.

Pulliam H. R. 1988. Sources, sinks, and population regulation. The American Naturalist **132**:652-661.

Pulliam, H. R., J. B. Dunning, Jr., and J. Liu. 1992. Population dynamics in complex landscapes: a case study. Ecological Applications **2**:165-177.

Ripple, W. J., G. A. Bradshaw, and T. A. Spies. 1991. Measuring forest landscape patterns in the Cascade range of Oregon, USA. Biological Conservation **57**:73-88.

Saunders D. A., R. J. Hobbs, and C. R. Margules. 1991. Biological consequences of ecosystem fragmentation: a review. Conservation Biology **5**:18-32.

Shaw, D. M., and S. F. Atkinson. 1990. An introduction to the use of geographic information systems for ornithological research. The Condor **92**:564-570.

Stamps, J. A., M. Buechner, and V. V. Krishnan. 1987. The effects of edge permeability and habitat geometry on emigration from patches of habitat. American Naturalist **129**:533-552.

Taylor, P. D., L. Fahrig, K Henein, and G. Merriam. 1993. Connectivity is a vital element of landscape structure. Oikos **68**:571-573.

Thomas, J. W., E. D. Forsman, J. B. Lint, E. C. Meslow, B. R. Noon, and J. Verner. 1990. A conservation strategy for the Northern Spotted Owl: report to the interagency scientific committee to address the conservation of the Northern Spotted Owl. United States Government Printing Office, Washington, D.C., USA.

Turner, M. G. 1989. Landscape Ecology: the effect of pattern on process. Annual Review of Ecology and Systematics **20**:171-197.

Turner, M. G., R. H. Gardner, V. H. Dale, and R. V. O'Neill. 1989*a*. Predicting the spread of disturbance across heterogeneous landscapes. Oikos **55**:121-129.

Turner, M. G., R. Costanza, and F. H. Sklar. 1989*b*. Methods to evaluate the performance of spatial simulation models. Ecological Modelling **48**:1-18.

Turner, M. G. 1990. Spatial and temporal analysis of landscape patterns. Landscape Ecology 4:21-30.

Turner, M. G., G. J. Arthaud, R. T. Engstrom, S. J. Hejl, J. Liu, S. Loeb, and K. McKelvey. *In press*. Usefulness of spatially explicit population models in land management.

USDA. U. S. Department of Agriculture. 1988. Final supplement to the environmental impact statement for an amendment to the Pacific Northwest Regional Guide. Two volumes. Portland, Oregon.

USDA. U.S. Department of Agriculture / U. S. Department of the Interior / National Oceanic and Atmospheric Administration / U. S. Environmental Protection Agency. 1993. Forest ecosystem management: an ecological, economic, and social assessment. Portland, Oregon: A report of the Forest Ecosystem Management Assessment Team.

USDA/USDI. U. S Department of Agriculture and U. S. Department of the Interior. 1994. Record of decision for amendments to Forest Service and Bureau of Land Management planning documents within the range of the northern spotted owl. Portland, Oregon: Forest Service, Bureau of Land Management. 74 pages plus appendix.

USDI. U. S. Department of the Interior. 1990. Endangered and threatened wildlife; determination of threatened status for the northern spotted owl; final rule. Fish and Wildlife Service, Portland, Oregon. 55 **Federal Register**, 26114-26194.

USDI. U. S. Department of the Interior. 1992. Final draft recovery plan for the northern spotted owl. Portland, Oregon: Fish and Wildlife Service. 2 volumes.

Whitcomb, R. F., J. F. Lynch, M. K. Klimkiewicz, C. S. Robbins, B. L. Whitcomb, and D. Bystrak. 1981. Effects of forest fragmentation on avifauna of the eastern deciduous forest. Pages 125-206 *in* R. L. Burgess and D. M. Sharpe, editors. Forest island dynamics in man-dominated landscapes. Springer-Verlag, New York, New York, USA.

Yahner R. H. 1988. Changes in wildlife communities near edges. Conservation Biology 2:333-339.

Appendix 1: Code From The Movement Module

This appendix includes some of the C-language code responsible for moving individual owls around within a GIS landscape. Many of the sub-routines referred to in this text have not been reproduced in this appendix. Annotation appears in italicized bold text to the right of the model code.

```
#include <stdio.h>
#include <xview/xview.h>
#include <group.h>
#include <math.h>
#include "Model.h"
#include "Include.h"

unsigned int          Dispersal_Width;
unsigned int          Dispersal_Height;

double                Dispersal_For_Ctr;
double                Dispersal_For_Rgt;
double                Dispersal_For_Lft;
double                Dispersal_Rev_Lft;
double                Dispersal_Rev_Rgt;
double                Dispersal_Rev_Ctr;

void
set_autocorrelation(correlation_level)
        int         correlation_level;
{
        double      alpha;
        double      beta;
        double      gamma;

        double      strict_forward;
        double      mostly_forward;
        double      mostly_reverse;
        double      strict_reverse;

        if (correlation_level == 0) {

                strict_forward = 1.0 / 6.0;
                mostly_forward = 1.0 / 6.0;
                mostly_reverse = 1.0 / 6.0;
                strict_reverse = 1.0 / 6.0;
```

This routine sets the probabilities that determine how linear the owls movements will be. The user supplies a single value between 0 and 100. At 0, there is no correlation with the previous move. At 100, the next move will always be made in the same direction as the previous move, if possible.

```
        } else if (correlation_level == 100) {

                strict_forward = 1.0;
                mostly_forward = 0.0;
                mostly_reverse = 0.0;
                strict_reverse = 0.0;

        } else {
                                                                        These probabilities were
                alpha = (double)correlation_level / 100.0;              simply chosen because they
                beta  = 1.0 - alpha;                                    work well, and not for any
                gamma = 1.0 + alpha;                                    other reason.

                mostly_forward = (1.0 / 6.0) * pow(beta, 1.0) * pow(gamma, 3.0);
                mostly_reverse = (1.0 / 6.0) * pow(beta, 3.0) * pow(gamma, 2.0);
                strict_reverse = (1.0 / 6.0) * pow(beta, 6.0) * pow(gamma, 1.0);

                strict_forward = 1.00 - 2.0 * mostly_forward - 2.0 * mostly_reverse - strict_reverse;

        }

        Dispersal_For_Ctr = strict_forward;                             Use the probabilities to
        Dispersal_For_Rgt = strict_forward + mostly_forward;            specify ranges within [0, 1].
        Dispersal_For_Lft = strict_forward + 2.0 * mostly_forward;
        Dispersal_Rev_Lft = strict_forward + 2.0 * mostly_forward + mostly_reverse;
        Dispersal_Rev_Rgt = strict_forward + 2.0 * mostly_forward + 2.0 * mostly_reverse;
        Dispersal_Rev_Ctr = 1.0;
}

int
get_random_neighbor(location)                                           This function returns the ID
        int         location;                                           of a neighboring hexagon. It
{                                                                       differs from the function
        int         next;                                               called get_hexagon_neighbor
        int         value;                                              only in that if the neighbor
```

```
        value = 6.0 * drand48();
        switch (value)
        {
                case 0: next = hexagon_index[location].TOP_LFT; break;
                case 1: next = hexagon_index[location].TOP_RGT; break;
                case 2: next = hexagon_index[location].CTR_LFT; break;
                case 3: next = hexagon_index[location].CTR_RGT; break;
                case 4: next = hexagon_index[location].BOT_LFT; break;
                case 5: next = hexagon_index[location].BOT_RGT; break;
        }

        return(next);
}
```

requested is absent because the current hexagon is on a boundary, then this routine returns a zero. In this situation, the function called get_hexagon_neighbor returns the ID of the hexagon that would be wrapped to, but only if the wrapping function has been turned on.

```
int
get_hexagon_neighbor(current, neighbor)
        int        current;
        int        neighbor;
{
        int        next;

        switch (neighbor)
        {
                case tl_hex: next = hexagon_index[current].TOP_LFT; break;
                case tr_hex: next = hexagon_index[current].TOP_RGT; break;
                case cl_hex: next = hexagon_index[current].CTR_LFT; break;
                case cr_hex: next = hexagon_index[current].CTR_RGT; break;
                case bl_hex: next = hexagon_index[current].BOT_LFT; break;
                case br_hex: next = hexagon_index[current].BOT_RGT; break;
        }

        if ((next == 0) && (Dispersal_Boundaries == wrapping)) next = hexagon_index[current].wrap;

        return(next);
}
```

This function returns the ID of the neighboring hexagon requested, or if necessary, the ID of the hexagon that would be wrapped to (but only if the wrapping function has been turned on).

```
void
draw_dispersal_path(start, finish)
        int         start;
        int         finish;
{
        int         row1, col1;
        int         row2, col2;

        row1 = (hexagon_index[start].row - zoom_top + 1) * zoom_magnification;
        col1 = (hexagon_index[start].col - zoom_lft + 1) * zoom_magnification;

        row2 = (hexagon_index[finish].row - zoom_top + 1) * zoom_magnification;
        col2 = (hexagon_index[finish].col - zoom_lft + 1) * zoom_magnification;

        XDrawLine(lndswin_dsp, Pixmap_2, lndswin_gc, col1, row1, col2, row2);
}
```

This function simply draws a line connecting the centers of the starting and finishing hexagons.

```
int
get_next_location(disperser, step, bias)
        int         disperser;
        int         step;
        int         bias;
{
        int         i;
        int         get_best_site;

        double      value;

        int         next;
        int         last;
        int         current;

        int         forward;
        int         for_rgt;
        int         rev_rgt;
```

This function makes the determination of which neighboring hexagon will be selected for an owls next move. This critical function has to take several issues into account. These are documented below. A suitable site, in the comments below, refers to an unoccupied suitable site for males, and for females refers to a single territorial male.

```
int        reverse;
int        rev_lft;
int        for_lft;

int        neighbors_index1[7];
int        neighbors_index2[7];
int        neighbors_scores[7];

Dispersal_Locations[0].bounce = no;

last    = Dispersal_Individuals[disperser].location1;             The last hexagon occupied.
current = Dispersal_Individuals[disperser].location0;             The hexagon now occupied.

if (step > Dispersal_Minimum_Steps) {                             Only applies to dispersers.

        if (Dispersal_Locations[current].suitable == yes) return(current);

        for (i = 0; i <= 6; ++i) neighbors_index1[i] = i;         If you are on a suitable site,
        randomize_int(neighbors_index1, 6);                       then take it. Otherwise check
                                                                  to see if any immediate
        for (i = 1; i <= 6; ++i) {                                neighbors are suitable.

                switch (neighbors_index1[i])
                {
                        case 1: next = get_hexagon_neighbor(current, tl_hex); break;
                        case 2: next = get_hexagon_neighbor(current, tr_hex); break;
                        case 3: next = get_hexagon_neighbor(current, cl_hex); break;
                        case 4: next = get_hexagon_neighbor(current, cr_hex); break;
                        case 5: next = get_hexagon_neighbor(current, bl_hex); break;
                        case 6: next = get_hexagon_neighbor(current, br_hex); break;
                }

                if (Dispersal_Locations[next].suitable == yes) {
                        if (next == hexagon_index[current].wrap)  Always erase the path if a
                                erase_dispersal_path(disperser);  wrapping event occurs.
                        return(next);
```

```
                }

            }

    }

    if (drand48() < (double)bias / 100.0) get_best_site = yes;     Determine if the bias to site
    else                                  get_best_site = no;      quality function will take
                                                                   precedence on this move. If it
    if (get_best_site == yes) {                                    does, then rank the neighbors.

            neighbors_scores[1] = hexagon_index[get_hexagon_neighbor(current, tl_hex)].score
                                  * 10.0 + 0.5;
            neighbors_scores[2] = hexagon_index[get_hexagon_neighbor(current, tr_hex)].score
                                  * 10.0 + 0.5;
            neighbors_scores[3] = hexagon_index[get_hexagon_neighbor(current, cl_hex)].score
                                  * 10.0 + 0.5;
            neighbors_scores[4] = hexagon_index[get_hexagon_neighbor(current, cr_hex)].score
                                  * 10.0 + 0.5;
            neighbors_scores[5] = hexagon_index[get_hexagon_neighbor(current, bl_hex)].score
                                  * 10.0 + 0.5;
            neighbors_scores[6] = hexagon_index[get_hexagon_neighbor(current, br_hex)].score
                                  * 10.0 + 0.5;
                                                                   Owls are not infinitely
            if ((neighbors_scores[1] == neighbors_scores[2]) &&    sensitive to quality, so let
                (neighbors_scores[2] == neighbors_scores[3]) &&    them notice only differences
                (neighbors_scores[3] == neighbors_scores[4]) &&    up to the first decimal place.
                (neighbors_scores[4] == neighbors_scores[5]) &&
                (neighbors_scores[5] == neighbors_scores[6])) get_best_site = no;

    }                                                              If the neighbors are all
                                                                   equivalent given this
    if (get_best_site == yes) {                                    criteria, then just move based
                                                                   on the previous direction.
            neighbors_index2[1] = 1;
            neighbors_index2[2] = 2;
```

```
        neighbors_index2[3] = 3;                                     This code just selects the
        neighbors_index2[4] = 4;                                     best neighbor so that it can
        neighbors_index2[5] = 5;                                     be moved to.
        neighbors_index2[6] = 6;

        sort_int(neighbors_scores, neighbors_index2, 6);

        for (i = 6; i >= 1; --i) {

                switch (neighbors_index2[i])
                {
                        case 1: next = get_hexagon_neighbor(current, tl_hex); break;
                        case 2: next = get_hexagon_neighbor(current, tr_hex); break;
                        case 3: next = get_hexagon_neighbor(current, cl_hex); break;
                        case 4: next = get_hexagon_neighbor(current, cr_hex); break;
                        case 5: next = get_hexagon_neighbor(current, bl_hex); break;
                        case 6: next = get_hexagon_neighbor(current, br_hex); break;
                }

                if (Dispersal_Locations[next].bounce == no) break;

        }                                                            The line above prohibits
                                                                     selecting a size that has been
} else {                                                             placed off limits (e.g. ocean
                                                                     or mountains).
        if (last == 0) {

                next = get_hexagon_neighbor(current, get_random_neighbor(current));

                forward = next;                                      This ELSE statement applies if
                for_rgt = next;                                      the move will NOT be simply to
                rev_rgt = next;                                      the best available neighbor.
                reverse = next;                                      The last == 0 case would be
                rev_lft = next;                                      true on the first move, or if
                for_lft = next;                                      the previous move was a wrap.
```

```
} else if (last == hexagon_index[current].TOP_LFT) {

        forward = hexagon_index[current].BOT_RGT;
        for_rgt = hexagon_index[current].BOT_LFT;
        rev_rgt = hexagon_index[current].CTR_LFT;
        reverse = hexagon_index[current].TOP_LFT;
        rev_lft = hexagon_index[current].TOP_RGT;
        for_lft = hexagon_index[current].CTR_RGT;

} else if (last == hexagon_index[current].TOP_RGT) {

        forward = hexagon_index[current].BOT_LFT;
        for_rgt = hexagon_index[current].CTR_LFT;
        rev_rgt = hexagon_index[current].TOP_LFT;
        reverse = hexagon_index[current].TOP_RGT;
        rev_lft = hexagon_index[current].CTR_RGT;
        for_lft = hexagon_index[current].BOT_RGT;

} else if (last == hexagon_index[current].CTR_LFT) {

        forward = hexagon_index[current].CTR_RGT;
        for_rgt = hexagon_index[current].BOT_RGT;
        rev_rgt = hexagon_index[current].BOT_LFT;
        reverse = hexagon_index[current].CTR_LFT;
        rev_lft = hexagon_index[current].TOP_LFT;
        for_lft = hexagon_index[current].TOP_RGT;

} else if (last == hexagon_index[current].CTR_RGT) {

        forward = hexagon_index[current].CTR_LFT;
        for_rgt = hexagon_index[current].TOP_LFT;
        rev_rgt = hexagon_index[current].TOP_RGT;
        reverse = hexagon_index[current].CTR_RGT;
        rev_lft = hexagon_index[current].BOT_RGT;
        for_lft = hexagon_index[current].BOT_LFT;
```

These statements exist to determine which of a hexagon's ACTUAL neighbors correspond to the six RELATIVE directions (forward, for_rgt, rev_rgt, reverse, rev_lft, for_lft). These relative directions are assigned probabilities by the set_autocorrelation function, based on the value of the linearity parameter.

```
        } else if (last == hexagon_index[current].BOT_LFT) {

                forward = hexagon_index[current].TOP_RGT;
                for_rgt = hexagon_index[current].CTR_RGT;
                rev_rgt = hexagon_index[current].BOT_RGT;
                reverse = hexagon_index[current].BOT_LFT;
                rev_lft = hexagon_index[current].CTR_LFT;
                for_lft = hexagon_index[current].TOP_LFT;

        } else if (last == hexagon_index[current].BOT_RGT) {

                forward = hexagon_index[current].TOP_LFT;
                for_rgt = hexagon_index[current].TOP_RGT;
                rev_rgt = hexagon_index[current].CTR_RGT;
                reverse = hexagon_index[current].BOT_RGT;
                rev_lft = hexagon_index[current].BOT_LFT;
                for_lft = hexagon_index[current].CTR_LFT;

        }

        value = drand48();

        if      (value < Dispersal_For_Ctr) next = forward;
        else if (value < Dispersal_For_Rgt) next = for_rgt;
        else if (value < Dispersal_For_Lft) next = for_lft;
        else if (value < Dispersal_Rev_Lft) next = rev_lft;
        else if (value < Dispersal_Rev_Rgt) next = rev_rgt;
        else if (value < Dispersal_Rev_Ctr) next = reverse;

}
```

Choose one of the six possible directions to go in, based on the value of the linearity parameter. It may still happen that the neighbor selected cannot be moved to, and the remainder of this function controls for such events.

This deals with attempts to move off of a data boundary.

```
if (next == 0) {
        if (Dispersal_Boundaries == absorbing) return(next);
        if (Dispersal_Boundaries == wrapping) next = hexagon_index[current].wrap;
}
```

```
        if (Dispersal_Locations[next].bounce == yes) next = 0;          If you hit a reflecting
                                                                        boundary, or site that cannot
        while (next == 0) {
                next = get_hexagon_neighbor(current, get_random_neighbor(current));
                if (Dispersal_Locations[next].bounce == yes) next = 0;
        }                                                               be moved to, then try again

        if (next == hexagon_index[current].wrap) erase_dispersal_path(disperser);

        return(next);                                                   via a recursive call. This
}                                                                       routine ONLY returns a zero if
                                                                        an owl hits an absorbing
void                                                                    boundary.
run_dispersal_model(avail_color, found_color, track_color, float_color)
        char        avail_color[];
        char        found_color[];                                      This is the main function that
        char        track_color[];                                      controls all movement events.
        char        float_color[];                                      It moves a collection of owls
{                                                                       until they have all elected to
        int         i, k;                                               stop for one reason or
        int         bias;                                               another. The demography module
        int         step;                                               calls this routine four times
        int         count;                                              per year to first conduct the
        int         disperser;                                          movements of both sexes of
        int         next_location;                                      floaters, and then for both
        int        *disperser_list;                                     sexes of dispersers.
        int         current_location;
        int         number_dispersers;

        double      prob_of_stopping;
        char        message[25];

        step  = 0;
        count = 1;

        number_dispersers = Dispersal_Number;
```

Just a bunch of preliminaries here. Note the comment lines.

```
Dispersal_Show_Movement = Show_Graphics;
Dispersal_Locations[0].floaters = 0;

Dispersal_Width  = xv_get(Patch_LndsWin->LndsWin, XV_WIDTH , NULL);
Dispersal_Height = xv_get(Patch_LndsWin->LndsWin, XV_HEIGHT, NULL);

if ((is_landscape_size_current() == no) || (lndswin_painted != yes))
        Dispersal_Show_Movement = no;

/* The number of steps a disperser will take is governed by the geometric distribution. */
/* The expected value of the geometric distribution is E(x) = (1 - p) / p, where p is   */
/* the probability of stopping during any one move. Solving this expression for p gives */
/* p = 1 / [E(x) + 1]. The user sets the value of E(x) (Dispersal_Expected_Path) & the   */
/* value of p -prob_of_stopping- is computed below. The probability of stopping is set  */
/* to zero if the dispersal path length is set to zero. The probability of stopping is  */
/* adjusted for the minimum path length as well. The minimum path length has to be less */
/* than the expected path length, otherwise the results would become trivial, or wrong. */

if ((Dispersal_Expected_Path > 0) &&
    (Dispersal_Minimum_Steps >= Dispersal_Expected_Path)) return;

if ((Dispersal_Minimum_Steps > 0) && (Dispersal_Maximum_Steps > 0) &&
    (Dispersal_Maximum_Steps <= Dispersal_Minimum_Steps)) return;

if (Dispersal_Expected_Path > 0)
        prob_of_stopping = 1.0 / (double)(Dispersal_Expected_Path -
                                          Dispersal_Minimum_Steps + 1);
else    prob_of_stopping = 0.0;

for (i = 0; i < Dispersal_Number; ++i)
        Dispersal_Individuals[i].location0 = Dispersal_Individuals[i].first_site;

if (Dispersal_Show_Movement == yes) {

        if (Hexagon_Overlay_Grid == yes)
                XCopyArea (lndswin_dsp, lndsgrid, Pixmap_1, lndswin_gc,
```

```
                      0, 0, Dispersal_Width, Dispersal_Height, 0, 0);
        else
                XCopyArea (lndswin_dsp, lndsmap, Pixmap_1, lndswin_gc,
                      0, 0, Dispersal_Width, Dispersal_Height, 0, 0);

        if (strcmp(avail_color, "OFF") != 0) {
                for (i = 1; i <= number_patches; ++i) {
                        if (Dispersal_Locations[i].suitable == yes) {
                                highlight_patch(i, avail_color, Pixmap_1);
                                draw_hexagon_edges(i, float_color);
                        }
                }
        }

        set_fgd_color(lndswin_dsp, lndswin_gc, lndswin_cmap, track_color);

}

bias = 0;
set_autocorrelation(100);
if (Initial_Linear_Steps < 0) Initial_Linear_Steps = 0;
```

Bias and linearity will be set to the appropriate values just below. This combination makes fledglings move in straight lines initially.

```
while (number_dispersers > 0) {
```

The module's principal loop.

```
        ++step;

        if (step == Initial_Linear_Steps + 1) {
                bias = Dispersal_Selectivity;
                set_autocorrelation(Dispersal_Autocorrelation);
        }

        if (Dispersal_Only == yes) {
                sprintf(message, "%s%d", "Step Number ", step);
                set_demography_footer(right, message);
        }
```

Turn on bias to quality and adjust linearity only AFTER dispersers have moved the minimum distance. This doesn't effect floaters because Initial_Linear_Steps = 0 for floaters.

```
if (Dispersal_Show_Movement == yes) {                     Part of the animation code.

        XCopyArea (lndswin_dsp, Pixmap_1, Pixmap_2, lndswin_gc,
                   0, 0, Dispersal_Width, Dispersal_Height, 0, 0);

        if ((Dispersal_Only == no) && (step == 1)) {
                for (i = 0; i < Dispersal_Number; ++i)
                        draw_hexagon_edges(Dispersal_Individuals[i].first_site,
                                           float_color);
                set_fgd_color(lndswin_dsp, lndswin_gc, lndswin_cmap, track_color);
        }

}

if (count > 0) {                                          True if any owls have stopped.

        if (number_dispersers < Dispersal_Number) free(disperser_list);
        disperser_list = (int *) calloc(number_dispersers + 1, sizeof(int));

        if (disperser_list == (int *)0) {
                printf("\nMemory Allocation Failure\n\n");
                exit(0);
        }                                                 If the number of moving owls
                                                          has changed, then rebuild the
        k = 0;                                            list that keeps track of them.

        for (i = 0; i < Dispersal_Number; ++i) {
                if (Dispersal_Individuals[i].status == searching)
                        disperser_list[++k] = i;
        }

}

randomize_int(disperser_list, number_dispersers);         Randomize the order in which
                                                          owls will take the next step.
count = 0;
```

```
                                                          This code actually moves the
for (i = 0; i < number_dispersers; ++i) {                 owls. Each owl that is still
                                                          moving takes a single step
        disperser = disperser_list[i+1];                  every time a pass is made
        ++Dispersal_Individuals[disperser].moves;         through the principal loop.

        current_location = Dispersal_Individuals[disperser].location0;
        next_location = get_next_location(disperser, step, bias);

        Dispersal_Individuals[disperser].location5 =      Record each owl's current
        Dispersal_Individuals[disperser].location4;       location, and its last five
        Dispersal_Individuals[disperser].location4 =      moves. This is done as part of
        Dispersal_Individuals[disperser].location3;       the linearity function, and as
        Dispersal_Individuals[disperser].location3 =      part of the process of
        Dispersal_Individuals[disperser].location2;       displaying the owl's movements
        Dispersal_Individuals[disperser].location2 =      on the computer screen. This
        Dispersal_Individuals[disperser].location1;       code updates those locations.
        Dispersal_Individuals[disperser].location1 =
        Dispersal_Individuals[disperser].location0;

        Dispersal_Individuals[disperser].location0 = next_location;
        Dispersal_Individuals[disperser].last_site = next_location;

        display_movement(disperser);                      The subroutine will bail out
                                                          if display is turned off.
        if (next_location == 0) {

                Dispersal_Individuals[disperser].status = failure;
                Dispersal_Individuals[disperser].last_site = current_location;

                ++count;                                  The next_location variable
                continue;                                 will only have been set to
                                                          zero if an owl has hit an
        }                                                 absorbing boundary.

        if (Dispersal_Locations[next_location].suitable == yes) {
```

```
        if (step < Dispersal_Minimum_Steps) continue;

        Dispersal_Locations[next_location].suitable = no;
        Dispersal_Individuals[disperser].status = success;

        if (Dispersal_Show_Movement == yes) {

            highlight_patch(next_location, found_color, Pixmap_1);
            draw_hexagon_edges(next_location, float_color);
            set_fgd_color(lndswin_dsp, lndswin_gc,
                   lndswin_cmap, track_color);

        }                                    If you have gotten this far,
                                             then you have found a suitable
        ++count;                             site and will settle there.
        continue;

    }
                                             If you get past this IF, then
    if (step == Dispersal_Maximum_Steps) {   you cannot move any further.

        Dispersal_Individuals[disperser].status = failure;
        ++Dispersal_Locations[next_location].floaters;

        if (Dispersal_Show_Movement == yes) {
            set_fgd_color(lndswin_dsp, lndswin_gc,
                   lndswin_cmap, float_color);
            draw_hexagon(next_location, mark, lndswin_dsp,
                  Pixmap_1, lndswin_gc);
            set_fgd_color(lndswin_dsp, lndswin_gc,
                   lndswin_cmap, track_color);
            flush_landscape_display();
        }

        ++count;
```

```
                continue;

        }
```

Decide here if an owl will elect to stop moving at this site and become a floater.

```
        if (drand48() < prob_of_stopping) {

                if (step < Dispersal_Minimum_Steps) continue;

                Dispersal_Individuals[disperser].status = failure;
                ++Dispersal_Locations[next_location].floaters;

                if (Dispersal_Show_Movement == yes) {
                        set_fgd_color(lndswin_dsp, lndswin_gc,
                                      lndswin_cmap, float_color);
                        draw_hexagon(next_location, mark, lndswin_dsp,
                                     Pixmap_1, lndswin_gc);
                        set_fgd_color(lndswin_dsp, lndswin_gc,
                                      lndswin_cmap, track_color);
                        flush_landscape_display();
                }

                ++count;
                continue;

        }

}
```

More animation code.

```
if (Dispersal_Show_Movement == yes) {
        XCopyArea (lndswin_dsp, Pixmap_2, lndswin_xid, lndswin_gc,
                   0, 0, Dispersal_Width, Dispersal_Height, 0, 0);
}

number_dispersers -= count;

if (number_dispersers == 0) {
```

```
                        free(disperser_list);
                        break;
                }

        }
```

More animation code.

```
        if (Dispersal_Show_Movement == yes) {
                XCopyArea (lndswin_dsp, Pixmap_1, lndswin_xid, lndswin_gc,
                           0, 0, Dispersal_Width, Dispersal_Height, 0, 0);
        }
}
```

Appendix 2: Code From The Spotted Owl Demography Module

This appendix includes a small portion of the C-language code responsible for tracking a population of spotted owls through time. Most of the sub-routines referred to in this text have not been reproduced in this appendix. Annotation appears in italicized bold text to the right of the model code.

```
#include <stdio.h>
#include <xview/xview.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <group.h>
#include <math.h>
#include "Model.h"
#include "Include.h"

int         Ind_First_Time = yes;
int         Ter_First_Time = yes;
int         S_F_First_Time = yes;

int         Survival_Index;

int         *Juv_Breed_Array;
int         *Sub_Breed_Array;
int         *Adt_Breed_Array;

int         *Juvenile_Array;
int         *Subadult_Array;
int         *Adult_Array;

int         *Floater_Array;
int         *Single_Array;
int         *Pair_Array;

float       *Juv_Surv_Array;
float       *Sub_Surv_Array;
float       *Adt_Surv_Array;

FILE        *Input_FilePointer;
FILE        *Output_FilePointer;

int         *Sampling_Array;
```

The variables on this page do not effect the basic spotted owl life history. They exist as part of the process of generating summary statistics that describe the outcome of a set of model runs.

```
void
movement(sex, fledglings, floaters)                                 This is the routine that calls
        int        sex;                                             the movement module. It is
        int        fledglings;                                      called four times per year
        int        floaters;                                        for: male floaters, female
{                                                                   floaters, male fledglings, and
        int        i;                                               female fledglings.
        int        k;
        int        key;
        int        self;
        int        site;
        int        mate;
        int        class;
        int        save_steps;
                                                                    Bail out if there are no owls
        if ((fledglings ==  no) && (floaters ==  no)) return;       that will need to be moved.
        if ((fledglings == yes) && (floaters == yes)) return;       Cannot move both fledglings
                                                                    and floaters at the same time.
        if (fledglings == yes) {
                if (sex == is_male) set_demography_footer(right, "Male Dispersal");
                else                set_demography_footer(right, "Female Dispersal");
        } else {
                if (sex == is_male) set_demography_footer(right, "Male Movement");
                else                set_demography_footer(right, "Female Movement");
        }

        Dispersal_Number = 0;

        for (i = 1; i <= Max_Numb; ++i) {                           Figure out how many owls are
                                                                    going to be moved.
                if (Individuals[i].sex != sex) continue;

                if ((Individuals[i].status != is_fledgling) &&
                    (Individuals[i].status != is_floater)) continue;
```

```
        if ((Individuals[i].status == is_fledgling) && (fledglings == no)) continue;
        if ((Individuals[i].status == is_floater  ) && (floaters   == no)) continue;

        ++Dispersal_Number;
}

if (Dispersal_Number == 0) return;

Dispersal_Locations   = (Dispersal_Loc_Struct *) calloc(Hexagon_Number_Hexagons + 1,
                                                        sizeof(Dispersal_Loc_Struct));
Dispersal_Individuals = (Dispersal_Ind_Struct *) calloc(Dispersal_Number,
                                                        sizeof(Dispersal_Ind_Struct));

if ((Dispersal_Locations == (Dispersal_Loc_Struct *)0) ||        Just building some requisite
    (Dispersal_Individuals == (Dispersal_Ind_Struct *)0)) {      arrays in this part.
        printf("\nMemory Allocation Failure\n\n");
        exit(0);
}
                                                                 The zero-element of the
Dispersal_Locations[0].bounce   = no;                            Dispersal_Locations array
Dispersal_Locations[0].suitable = no;                            should never be accessed, but
Dispersal_Locations[0].floaters = 0;                             this is just to be safe.

for (i = 1; i <= Hexagon_Number_Hexagons; ++i) {                 Set up the movement arrays.

        Dispersal_Locations[i].bounce   = Territories[i].bounce;
        Dispersal_Locations[i].floaters = Territories[i].floaters;

        if (sex == is_male) Dispersal_Locations[i].suitable = hexagon_index[i].territory;
        else                Dispersal_Locations[i].suitable = no;

        if (Dispersal_Locations[i].bounce == yes) Dispersal_Locations[i].suitable = no;

}

for (i = 1; i <= Max_Numb; ++i) {
```

```
                                                                 Identify sites with single
        if (sex == is_female) {                                  territorial males for females
                if ((Individuals[i].sex == is_male) &&           to pair with.
                    (Individuals[i].status == is_territorial))
                        Dispersal_Locations[Individuals[i].site].suitable = yes;
                continue;
        }

        if ((Individuals[i].status == is_paired) ||              Identify empty suitable sites
            ((Individuals[i].status == is_territorial) &&        for males to settle into.
            (Individuals[i].sex == is_male)))
                Dispersal_Locations[Individuals[i].site].suitable = no;

}

k = 0;

for (i = 1; i <= Max_Numb; ++i) {                                This FOR loop sets up the
                                                                 Dispersal_Individuals array,
        if (Individuals[i].sex != sex) continue;                 and accomplishes a few other
                                                                 tasks.
        if ((Individuals[i].status != is_fledgling) &&
            (Individuals[i].status != is_floater)) continue;     Only pick the correct owls.

        if ((Individuals[i].status == is_fledgling) && (fledglings == no)) continue;
        if ((Individuals[i].status == is_floater  ) && (floaters   == no)) continue;

        key   = Individuals[i].key;
        site  = Individuals[i].site;
        class = Individuals[i].class;

        Dispersal_Individuals[k].key        = key;               Fill the movement structure's
        Dispersal_Individuals[k].class      = class;             elements with the values
        Dispersal_Individuals[k].moves      = 0;                 necessary to correctly
        Dispersal_Individuals[k].status     = searching;         initiate the movement process.
        Dispersal_Individuals[k].first_site = site;
```

```
        Dispersal_Individuals[k].last_site  = 0;
        Dispersal_Individuals[k].location0  = 0;
        Dispersal_Individuals[k].location1  = 0;
        Dispersal_Individuals[k].location2  = 0;
        Dispersal_Individuals[k].location3  = 0;
        Dispersal_Individuals[k].location4  = 0;
        Dispersal_Individuals[k].location5  = 0;

        if (Individuals[i].status == is_floater) --Dispersal_Locations[site].floaters;

        remove_individual(i);                              Remove owls to be in motion.
                                                           They'll be put back later.
        if ((Show_Graphics == yes) && (floaters == yes)) {
                set_fgd_color(lndswin_dsp, lndswin_gc, lndswin_cmap, Searcher_Color);
                draw_hexagon(site, mark, lndswin_dsp, lndswin_xid, lndswin_gc);
        }

        if (++k == Dispersal_Number) break;                Break out when you've dealt
                                                           with all of the owls that will
}                                                          move this time.

if (Show_Graphics == yes) copy_landsgrid_image();

save_steps = Dispersal_Minimum_Steps;
                                                           This is where the minimum
if (floaters == yes) {                                     distance requirement gets
        Initial_Linear_Steps = 2;                          turned on for fledglings, and
        Dispersal_Minimum_Steps = Initial_Linear_Steps;    turned off for floaters.
} else  Initial_Linear_Steps = 0;

run_dispersal_model("OFF", Searcher_Color, Searcher_Color, Floater_Color);

if (fledglings == yes) get_movement_distances();           Call the movement model and
Dispersal_Minimum_Steps = save_steps;                      tally up the results.

for (i = 0; i < Dispersal_Number; ++i) Dispersal_Individuals[i].moves;
```

Put back the owls that did not find a suitable site. They are added back into the population as floaters.

```
for (i = 0; i < Dispersal_Number; ++i) {

        if (Dispersal_Individuals[i].status == failure) {

                key   = Dispersal_Individuals[i].key;
                site  = Dispersal_Individuals[i].last_site;
                class = Dispersal_Individuals[i].class;

                add_individual(key, sex, site, 0, class, is_floater);

        }

}
```

Put back the owls that did find a suitable site. If a pair has been created, then take the appropriate actions.

Note that Next_One is just the ID of the next owl that will be added to the population.

```
for (i = 0; i < Dispersal_Number; ++i) {

        if (Dispersal_Individuals[i].status == success) {

                self  = Next_One;
                key   = Dispersal_Individuals[i].key;
                site  = Dispersal_Individuals[i].last_site;
                class = Dispersal_Individuals[i].class;

                if (sex == is_male) mate = Territories[site].female;
                else                mate = Territories[site].male;

                add_individual(key, sex, site, mate, class, is_territorial);
                if (mate > 0) make_pair(self, mate, site);
                display_individual(self);

        }

}

free(Dispersal_Locations);
```

```
        free(Dispersal_Individuals);

        set_demography_footer(right, "\0");
}
```

```
void
add_new_fledglings(territory_index, number_breeding, number_fledging)
        int         *territory_index;
        int          number_breeding;
        int          number_fledging;
{
        int         i;
        int         index;
        int         count;
        int         brood_size;

        int         sex;
        int         site;

        double         break_point;
        double         random_number;

        if (number_fledging <= 0) return;
```

This function assigns new fledglings to breeding owl pairs. This is called by the reproduction sub-routine. The overall number of fledglings to be apportioned out between all of the breeding pairs is determined by one of the following functions: get_reproductive_adults, get_reproductive_subadults, or get_reproductive_juveniles.

```
        count = 0;

        randomize_int(territory_index, number_breeding);
```

Randomize the list of owl pairs that will receive the new fledglings.

```
        for (index = 1; index <= number_breeding; ++index) {
```

Loop through each of the breeding owl pairs.

```
                break_point = 0;
                random_number = drand48();

                site  = Individuals[territory_index[index]].site;

                if (count < number_fledging) {
```

This random number will fix the brood size for this nest. The determination of this brood size will be conducted by code that comes next.

```
                        for (brood_size = 1; brood_size <= Largest_Brood; ++brood_size) {

                                break_point += (double)Brood_Size_Probs[brood_size] / 100.0;
                                if (random_number < break_point) break;
                                                                          Break out when the correct
                        }                                                 brood size is identified.

                        for (i = 0; i < brood_size; ++i) {                Now put out that many owls.

                                if (drand48() < (double)Percent_Females / 100.0) sex = is_female;
                                else                                             sex = is_male;

                                add_individual(++KEY, sex, site, 0, is_juvenile, is_fledgling);

                                if (++count == number_fledging) break;    Bail out if all the available
                                                                          fledglings are used up.
                        }
                                                                          This routine does NOT check to
                }                                                         see if too few fledglings are
                                                                          put out, but this isn't a
        }                                                                 problem with a species that
}                                                                         has a low fecundity (e.g. the
                                                                          spotted owl).
int *
get_reproductive_adults()                                                 This function determines which
{                                                                         suitable sites have breeding
        int         i, k;                                                 pairs of owls in them, and it
        int        *list;                                                 builds a list of these sites.
                                                                          Such a pair has to include an
        Adults_Breeding = 0;                                              ADULT female to qualify.

        for (i = 1; i <= Max_Numb; ++i) {
                if ((Individuals[i].status == is_paired) &&
                    (Individuals[i].class  == is_adult ) &&
                    (Individuals[i].sex    == is_female)) ++Adults_Breeding;
```

```
        }

        if (Adults_Breeding == 0) return(0);

        list = (int *) calloc(Adults_Breeding + 1, sizeof(int));

        if (list == (int *)0) {
                printf("\nMemory Allocation Failure\n\n");
                exit(0);
        }

        k = 0;

        for (i = 1; i <= Max_Numb; ++i) {
                if ((Individuals[i].status == is_paired) &&
                    (Individuals[i].class  == is_adult ) &&
                    (Individuals[i].sex    == is_female)) list[++k] = i;
        }

        return(list);
}
```

This function determines which suitable sites have breeding pairs of owls in them, and it builds a list of these sites. Such a pair has to include a SUBADULT female to qualify.

```
int *
get_reproductive_subadults()
{
        int         i, k;
        int        *list;

        Subadults_Breeding = 0;

        for (i = 1; i <= Max_Numb; ++i) {
                if ((Individuals[i].status == is_paired   ) &&
                    (Individuals[i].class  == is_subadult) &&
                    (Individuals[i].sex    == is_female  )) ++Subadults_Breeding;
        }
```

```
        if (Subadults_Breeding == 0) return(0);

        list = (int *) calloc(Subadults_Breeding + 1, sizeof(int));

        if (list == (int *)0) {
                printf("\nMemory Allocation Failure\n\n");
                exit(0);
        }

        k = 0;

        for (i = 1; i <= Max_Numb; ++i) {
                if ((Individuals[i].status == is_paired  ) &&
                    (Individuals[i].class  == is_subadult) &&
                    (Individuals[i].sex    == is_female  )) list[++k] = i;
        }

        return(list);
}
```

This function determines which suitable sites have breeding pairs of owls in them, and it builds a list of these sites. Such a pair has to include a JUVENILE female to qualify.

```
int *
get_reproductive_juveniles()
{
        int         i, k;
        int        *list;

        Juveniles_Breeding = 0;

        for (i = 1; i <= Max_Numb; ++i) {
                if ((Individuals[i].status == is_paired  ) &&
                    (Individuals[i].class  == is_juvenile) &&
                    (Individuals[i].sex    == is_female  )) ++Juveniles_Breeding;
        }

        if (Juveniles_Breeding == 0) return(0);
```

```
        list = (int *) calloc(Juveniles_Breeding + 1, sizeof(int));

        if (list == (int *)0) {
                printf("\nMemory Allocation Failure\n\n");
                exit(0);
        }

        k = 0;

        for (i = 1; i <= Max_Numb; ++i) {
                if ((Individuals[i].status == is_paired  ) &&
                    (Individuals[i].class  == is_juvenile) &&
                    (Individuals[i].sex    == is_female  )) list[++k] = i;
        }

        return(list);
}

void
reproduction()
{                                                                       This function creates the
        int          i;                                                 reproductive pulse for a
        int          number_fledging;                                   single year. It calls all of
        int         *territory_index;                                   the previous functions in
                                                                        order to accomplish this task.
        set_demography_footer(right, "Reproduction");                   Ageing is also handled here.

        territory_index = get_reproductive_adults();                    Take care of the reproducing
                                                                        adult females.
        if (territory_index != (int *)0) {

                number_fledging = (int)(Adult_Fecundity * (float)Adults_Breeding + 0.5);
                add_new_fledglings(territory_index, Adults_Breeding, number_fledging);
                free(territory_index);

        }
```

```
territory_index = get_reproductive_subadults();                Take care of the reproducing
                                                               subadult females.
if (territory_index != (int *)0) {

        number_fledging = (int)(Subadult_Fecundity * (float)Subadults_Breeding + 0.5);
        add_new_fledglings(territory_index, Subadults_Breeding, number_fledging);
        free(territory_index);

}

for (i = 1; i <= Max_Numb; ++i) {                              This is where the subadults
                                                               grow up to become adults.
        if (Individuals[i].class != is_subadult) continue;
        Individuals[i].class = is_adult;
        --Number_Subadults;
        ++Number_Adults;

}

territory_index = get_reproductive_juveniles();                Take care of the reproducing
                                                               juvenile females.
if (territory_index != (int *)0) {

        number_fledging = (int)(Juvenile_Fecundity * (float)Juveniles_Breeding + 0.5);
        add_new_fledglings(territory_index, Juveniles_Breeding, number_fledging);
        free(territory_index);

}

for (i = 1; i <= Max_Numb; ++i) {                              This is where the juveniles
                                                               grow up to become subadults.
        if (Individuals[i].class  != is_juvenile)  continue;
        if (Individuals[i].status == is_fledgling) continue;
        Individuals[i].class = is_subadult;
        --Number_Juveniles;
```

```
                ++Number_Subadults;

        }

        set_demography_footer(right, "\0");
}

float
linear_interpolation(score, min_score, max_score, min_param, max_param)
        float           score;
        float           min_score, max_score;               This function accomplishes the
        float           min_param, max_param;               linear interpolation that is
{                                                           necessary to determine
        float           result;                             survival rates and site
                                                            fidelities in sites having
        if ((min_score == max_score) || (min_param == max_param)) {    qualities in-between the
                                                            minimum and maximum values.
                result = (min_param + max_param) / 2.0;
                return(result);

        }

        result = min_param + (score - min_score) * (max_param - min_param) / (max_score - min_score);
        return(result);
}

float
logistic_interpolation(score, min_score, max_score, min_param, max_param)
        float           score;
        float           min_score, max_score;               This function accomplishes the
        float           min_param, max_param;               logistic interpolation that is
{                                                           necessary to determine
        float           x, y;                               survival rates and site
        float           result;                             fidelities in sites having
                                                            qualities in-between the
        if ((min_score == max_score) || (min_param == max_param)) {    minimum and maximum values.
```

```
                result = (min_param + max_param) / 2.0;
                return(result);

        }

        x = 62.0 * (score - min_score) / (max_score - min_score);
        y = (float)(1.0 / (1.0 + 500.0 * exp((double)(-x / 4.988))));

        result = min_param + y * (max_param - min_param);
        return(result);
}
```

This produces a logistic curve that is symmetric about it's inflection point, and is rescaled so that it's y-value ranges between zero and one.

```
float
interpolate(score, min_score, max_score, min_param, max_param)
        float           score;
        float           min_score, max_score;
        float           min_param, max_param;
{
        float           result;

        if (Fit_To_Quality == Linear)
                result = linear_interpolation(score, min_score, max_score, min_param, max_param);
        else    result = logistic_interpolation(score, min_score, max_score, min_param, max_param);

        return(result);
}
```

This function exists only to select and call the appropriate interpolation function, based on the value of a switch set by the user.

```
void
recruit_floaters()
{
        int             i;
        int             sex;
        int             site;

        float           score;
```

Single territorial male owls in sites of marginal quality may elect to move on in hopes of obtaining a superior site. This function controls an owls decision regarding whether to keep its present site or move.

```
float       stay_on_site;
float       min_fidelity;
float       max_fidelity;

char        color[50];

for (i = 1; i <= Max_Numb; ++i) {                           Go through all of the owls
                                                            looking for single territorial
        if (Individuals[i].status != is_territorial) continue;   males.

        sex = Individuals[i].sex;
        site = Individuals[i].site;
        score = hexagon_index[site].score;
                                                            Set the cutoff value that will
        if (sex == is_male) {                               be used to make the decision.

                min_fidelity = (float)Site_Fidelity_Min / 100.0;
                max_fidelity = (float)Site_Fidelity_Max / 100.0;

                stay_on_site = interpolate(score, Min_Terr_Score, Max_Terr_Score,
                                           min_fidelity, max_fidelity);

        } else       stay_on_site = 0;                      Females have to move, but this
                                                            would only be necessary if a
        if (drand48() >= (double)stay_on_site) {            paired female's mate died.

                --Number_Singles;                           This IF statement is entered
                ++Number_Floaters;                          if an individual HAS decided
                ++Territories[site].floaters;               to move on and look for a
                Individuals[i].status = is_floater;         better site.

                if (sex == is_male) Territories[site].male   = 0;
                else                Territories[site].female = 0;

                if (Show_Graphics == yes) {
                        get_landscape_color(site, color);
```

```
                                highlight_patch(site, color, lndswin_xid);
                                update_hexagon_edges(site);
                        }

                }

        }
}

void
overwinter()
{
        int        i;
        int        site;
        int        class;

        int        start_juv;
        int        start_sub;
        int        start_adt;

        float        score;
        float        survival;
        float        min_survival;
        float        max_survival;
```

This is the demography module's survival function.

```
        set_demography_footer(right, "Survival");

        if (Current_Year > 2) {
                start_juv = Number_Juveniles;
                start_sub = Number_Subadults;
                start_adt = Number_Adults;
        } else {
                start_juv = 0;
                start_sub = 0;
                start_adt = 0;
        }
```

This code exists to track the effective survival rates of the owls. Since a landscape is initialized only with pairs, you have to wait two years to build up populations of the other age classes of owls before the averaging process can be started.

```
for (i = 1; i <= Max_Numb; ++i) {                              The main loop controlling the
                                                               survival process.
        site   = Individuals[i].site;
        class  = Individuals[i].class;

        score = (double)hexagon_index[site].score;

        if (class == is_juvenile) {                            The fate of a juvenile owl is
                                                               decided here.
                min_survival = Juvenile_Min_Survival;
                max_survival = Juvenile_Max_Survival;

                survival = interpolate(score, Min_Site_Score, Max_Site_Score,
                                       min_survival, max_survival);

                if (drand48() >= (double)survival) remove_individual(i);

        } else if (class == is_subadult) {                     The fate of a subadult owl is
                                                               decided here.
                min_survival = Subadult_Min_Survival;
                max_survival = Subadult_Max_Survival;

                survival = interpolate(score, Min_Site_Score, Max_Site_Score,
                                       min_survival, max_survival);

                if (drand48() >= (double)survival) remove_individual(i);

        } else if (class == is_adult) {                        The fate of an adult owl is
                                                               decided here.
                min_survival = Adult_Min_Survival;
                max_survival = Adult_Max_Survival;

                survival = interpolate(score, Min_Site_Score, Max_Site_Score,
                                       min_survival, max_survival);
```

```
                        if (drand48() >= (double)survival) remove_individual(i);

                }

        }                                                                   Compute the effective juvenile
                                                                            survival rate for this year.
        if (start_juv < 1) {                                                The -999 is a flag that says
                Juvenile_True_Survival = -999.0;                            no value exists for a year.
        } else {
                Juvenile_True_Survival   = (float)Number_Juveniles / (float)start_juv;
                Juvenile_Survival_Count += Number_Juveniles;
                Juvenile_Survival_Total += start_juv;
        }                                                                   Compute the effective subadult
                                                                            survival rate for this year.
        if (start_sub < 1) {                                                The -999 is a flag that says
                Subadult_True_Survival = -999.0;                            no value exists for a year.
        } else {
                Subadult_True_Survival   = (float)Number_Subadults / (float)start_sub;
                Subadult_Survival_Count += Number_Subadults;
                Subadult_Survival_Total += start_sub;
        }                                                                   Compute the effective adult
                                                                            survival rate for this year.
        if (start_adt < 1) {                                                The -999 is a flag that says
                Adult_True_Survival = -999.0;                               no value exists for a year.
        } else {
                Adult_True_Survival   = (float)Number_Adults / (float)start_adt;
                Adult_Survival_Count += Number_Adults;
                Adult_Survival_Total += start_adt;
        }

        set_demography_footer(right, "\0");
}

void
life_history()                                                              This is the main routine that
{                                                                           propagates a population of
```

owls forward in time for one year. It is self explanatory.

```
        int         floaters;
        int         fledglings;

        overwinter();

        recruit_floaters();

        fledglings = no;
        floaters   = yes;

        movement(is_male  , fledglings, floaters);
        movement(is_female, fledglings, floaters);

        reproduction();

        fledglings = yes;
        floaters   = no;

        movement(is_male  , fledglings, floaters);
        movement(is_female, fledglings, floaters);

        census();
}
```

This routine conducts a single replicate run of the spotted owl demography module. It is called by the function named run_demography_model, which appears below. The current function must to do all of the work associated with loading in new landscapes, but most of the consistency checking is done by its calling routine.

```
int
do_demography()
{
        int         display;
        int         last_year;

        display = no;

        if (Current_Year == 0) {

                TS_Year = Current_Year;
                update_time_series();
```

```
                                                                  Get everything started here.
        display_year();
        if (update_landscape(display) == failure) return(failure);
        initialize_model();

        census();                                                 Perform the year-zero census.
        ++Current_Year;                                           Now you are at year = 1.

}

display = yes;

if (Show_Graphics == yes) {
        if ((Current_Year == 1) && (Time_Series[0].toggle == on)) display_all_individuals();
        else display_everyone();
}

last_year = Current_Year + Number_Years;                          Determine when to quit.

while (Current_Year < last_year) {                                Loop through all of the years.

        Fledge_Net_Distance = 0;                                  Get ready to compute a bunch
        Fledge_Sum_Distance = 0;                                  of summary information.

        TS_Year = Current_Year;                                   Get the year-one landscape.
        update_time_series();

        display_year();                                           Bail out if all owls are dead.

        if (Number_Juveniles + Number_Subadults + Number_Adults == 0) break;
        if (update_landscape(display) == failure) return(failure);

        get_sf_values();                                          Read in some parameter values.
        life_history();                                           Make the owls do their thing
                                                                  for a single year.

        ++Current_Year;
```

```
                clean_array();                                                  Condense the principle data
                check_array();                                                  array, and then check it for
                                                                                errors. The checking is very
        }                                                                       thorough.

        if (Show_Graphics == yes) copy_landsgrid_image();
        return(success);
}

void
run_demography_model()                                                          This is the function that is
{                                                                               actually called when the user
        int             i;                                                      pushes the button to run the
        int             count;                                                  model. It is responsible for
        char            message[50];                                            making sure the user didn't do
        int             old_display_value;                                      anything illegal. It is also
        int             old_graphics_value;                                     where the code resides that
                                                                                generates multiple replicate
        Dispersal_Only = no;                                                    model runs. Much of the
                                                                                generation of summary data
        set_demography_footer(left , "\0");                                     takes place here as well.
        set_demography_footer(right, "\0");

        if (is_landscape_present() == no) {                                     Must have a landscape to run
                set_demography_footer(left , "Landscape Not Present");          the model.
                return;
        }

        if (Hexagon_Identifier != Landscape_Identifier) {                       The current landscape must in
                set_demography_footer(left , "Territories Not Present");        fact be a territory map.
                return;
        }

        if (Number_SF_Values < 1) {
                set_demography_footer(left , "S.F. Data Not Present");          Need the appropriate data.
```

```
        return;                                                   Build some arrays if they are
}                                                                 not already there for you.

if (Demography_Identifier != Landscape_Identifier) initialize_territories();

old_display_value = zoom_auto_display;                            Make sure the hexagon size
zoom_auto_display = off;                                          displayed is the hexagon size
erase_zoomwin();                                                  used. Turn off the zoom window
restore_hexagon_size();                                           display to save time.

old_graphics_value = Show_Graphics;                               Turn off display if not ok.

if ((is_landscape_size_current() == no) ||
    (Hexagon_Size != Hexagon_Save_Size) || (lndswin_painted != yes)) {
        Show_Graphics = no;
        xv_set(Demography_ParameterWin->DisplayOn , PANEL_VALUE, 0, NULL);
        xv_set(Demography_ParameterWin->DisplayOff, PANEL_VALUE, 1, NULL);
}

set_demography_footer(right, "Reading Time Series");

count = 0;

for (TS_Year = 0; TS_Year <= Time_Series_Length; ++TS_Year) {    Check the time series data.
                                                                  This means reading the headers
        if (Time_Series[TS_Year].toggle == off) continue;         of all the files associated
        update_time_series();                                     with each landscape to be used
        ++count;                                                  at any point in the run. Bail
                                                                  out if they are not all ok.
        if (check_time_series() == failure) {
                set_demography_footer(left, "Time Series Error");
                sprintf(message, "%s%d", "Year : ", TS_Year);
                set_demography_footer(right, message);
                return;
        }
```

```
}

if ((count > 0) && (Number_Runs > 1) && (Time_Series[0].toggle == off)) {
        set_demography_footer(left, "Time Series Error");
        set_demography_footer(right, "Year Zero Not Set");       Multiple replicates have to
        return;                                                   specify a year-zero landscape
}                                                                 so that a new replicate does
                                                                  not start of with a territory
TS_Year = 0;                                                      map that was in use at the end
update_time_series();                                             of the previous replicate.

Fledge_Distance_Count = 0;
Fledge_Net_Total      = 0;
Fledge_Sum_Total      = 0;
                                                                  This code makes up part of the
Adult_Survival_Count    = 0;                                      process of generating the
Subadult_Survival_Count = 0;                                      summary statistics that are
Juvenile_Survival_Count = 0;                                      presented to the user at the
                                                                  end of the model runs.
Adult_Survival_Total    = 0;
Subadult_Survival_Total = 0;
Juvenile_Survival_Total = 0;

Juv_Surv_Array = (float *) calloc(Number_Runs * (Number_Years + 1), sizeof(float));
Sub_Surv_Array = (float *) calloc(Number_Runs * (Number_Years + 1), sizeof(float));
Adt_Surv_Array = (float *) calloc(Number_Runs * (Number_Years + 1), sizeof(float));

if ((Juv_Surv_Array == (float *)0) ||
    (Sub_Surv_Array == (float *)0) ||
    (Adt_Surv_Array == (float *)0)) {
        printf("\nMemory Allocation Failure\n\n");
        exit(0);
}

for (i = 0; i < Number_Runs * (Number_Years + 1); ++i) {         I just use -999 as a flag that
        Juv_Surv_Array[i] = -999;                                 this array entry is not valid.
```

```
        Sub_Surv_Array[i] = -999;
        Adt_Surv_Array[i] = -999;
}                                                                 A KEY is an identifier that
                                                                  uniquely specifies an owl,
KEY = 0;                                                          even given multiple replicate
                                                                  runs. It is not used by the
Survival_Index = 0;                                               model, but could be used to
get_range_of_scores();                                            follow individual movements.

for (i = 0; i <= Hexagon_Number_Hexagons; ++i) Territories[i].occupancy = 0;

if (Do_Sampling == yes) {
        Sampling_Array = (int *) calloc ((Number_Years + 1) * (Hexagon_Number_Hexagons + 1),
                                         sizeof(int));
}                                                                 Get ready to track occupancy
                                                                  rates, and to build the
set_demography_footer(left , "\0");                               sampling array.
set_demography_footer(right, "\0");

if (Number_Runs == 1) {                                           Things are simple if only one
                                                                  replicate is being performed.
        if (Current_Year == 0) write_demographic_parameters();    Then there will be less
        else                   update_individuals_status();       summary data to generate.

        Current_Run = 1;

        if (get_output_file() == failure) return;
        if (do_demography()   == failure) return;

        print_bot_header();

} else {                                                          Multiple replicates go here.

        Juvenile_Array  = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));
        Subadult_Array  = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));
        Adult_Array     = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));
```

```
Floater_Array   = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));
Single_Array    = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));
Pair_Array      = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));

Juv_Breed_Array = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));
Sub_Breed_Array = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));
Adt_Breed_Array = (int *) calloc(Number_Runs * (Number_Years + 1), sizeof(int));

if ((Juvenile_Array  == (int *)0) ||
    (Subadult_Array  == (int *)0) ||         This code is used to generate
    (Adult_Array     == (int *)0) ||         means and standard deviations
    (Floater_Array   == (int *)0) ||         of the different population
    (Single_Array    == (int *)0) ||         measures, on a yearly basis.
    (Pair_Array      == (int *)0) ||         These data cannot be tabulated
    (Juv_Breed_Array == (int *)0) ||         unless multiple replicates are
    (Sub_Breed_Array == (int *)0) ||         conducted.
    (Adt_Breed_Array == (int *)0)) {
        printf("\nMemory Allocation Failure\n\n");
        exit(0);
}

write_demographic_parameters();                Normal I/O and bookkeeping.

Current_Run = 1;

if (get_output_file() == failure) return;

while (Current_Run <= Number_Runs) {           Crank through all of the
        if (do_demography() == failure) return;  replicate model runs here.
        ++Current_Run;
        Current_Year = 0;
}

print_summaries();                             Write out the summary data.
```

```
        free(Juvenile_Array);
        free(Subadult_Array);
        free(Adult_Array);
        free(Floater_Array);
        free(Single_Array);
        free(Pair_Array);
        free(Juv_Breed_Array);
        free(Sub_Breed_Array);
        free(Adt_Breed_Array);

    }

    free(Juv_Surv_Array);
    free(Sub_Surv_Array);
    free(Adt_Surv_Array);

    if (Do_Sampling == yes) {
        print_sampling_data();
        free(Sampling_Array);
    }

    Current_Run = 0;
    if (Output_FilePointer != stdout) fclose(Output_FilePointer);
    Show_Graphics     = old_graphics_value;
    zoom_auto_display = old_display_value;

    if (Show_Graphics == yes) {
        xv_set(Demography_ParameterWin->DisplayOn , PANEL_VALUE, 1, NULL);
        xv_set(Demography_ParameterWin->DisplayOff, PANEL_VALUE, 0, NULL);
    } else {
        xv_set(Demography_ParameterWin->DisplayOn , PANEL_VALUE, 0, NULL);
        xv_set(Demography_ParameterWin->DisplayOff, PANEL_VALUE, 1, NULL);
    }

    set_demography_footer(right, "\0");
}
```

More bookkeeping and code devoted to graphics.

CURRICULUM VITAE

Nathan H. Schumaker
Center for Streamside Studies AR-10
University of Washington
Seattle, WA 98195
(206) 685-8658

Education

B.S. in Physics (1985)
University Of California, Santa Cruz, California

B.S. in Mathematics (1985)
University Of California, Santa Cruz, California

M.S. in Applied Mathematics (1989)
University Of Washington, Seattle, Washington

Ph.D. in Forest Ecosystem Analysis (1995)
University Of Washington, Seattle, Washington

Professional Employment

4/94 to 1/95

College of Forest Resources
University of Washington, Seattle, Washington.
Designed a spatially explicit, individual-based, life history simulator for the Northern Spotted Owl. Conducted a viability analysis for the State of Washington Department of Natural Resources that examined Northern Spotted Owls on Washington State forest lands.

9/93 to 4/94

Environmental Studies Board
University of California, Santa Cruz, California.
Designed, set up, and managed a GIS facility for the Environmental Studies Board of the University of California at Santa Cruz.

1/91 to 6/92

College of Forest Resources
University of Washington, Seattle, Washington.
Designed, set up, and maintained the College of Forestry GIS laboratory at the University of Washington.

7/90 to Present

Center for Streamside Studies
University of Washington, Seattle, Washington.
Research Assistant - Led a research effort to assess the value of GIS data to population viability models. Applied both mathematical and statistical models, as well as computer simulations, to the investigation of ecological consequences of changing landscape patterns.

7/88 to 6/90

Center for Quantitative Science in Forestry, Fisheries, and Wildlife
University of Washington, Seattle, Washington.
Research Assistant - Constructed a computer simulation model for the downstream migration of Columbia River salmon stocks.

10/87 to 6/88

Applied Mathematics Department
University of Washington, Seattle, Washington.
Teaching Assistant for three upper-division courses in ordinary and partial differential equations.

10/85 to 9/87

The Santa Cruz Operation
Santa Cruz, California.
Technical support analyst, quality assurance technician, and database manager.

7/84 to 6/85

Santa Cruz Institute for Particle Physics
University of California, Santa Cruz, California.
Laboratory technician. Worked at the Santa Cruz Institute for Particle Physics and at the Stanford Linear Accelerator Center.

Publications

Lee, R. G., R. Flamm, M. G. Turner, C. Bledsoe, P. Chandler, C. DeFerrari, R. Gottfried, R. J. Naiman, N. Schumaker, and D. Wear. 1992. Integrating sustainable development and environmental vitality: a landscape ecology approach. Pages 497-519 *in* R. J. Naiman, editor. Watershed management: balancing sustainability and environmental change. Springer-Verlag, New York, New York, USA.

Deutschman, D. H., G. A. Bradshaw, W. M. Childress, K. L. Daly, D. Grünbaum, M. Pascual, N. H. Schumaker, and J. Wu. 1993. Mechanisms Of Patch Formation. Pages 184-209 *in* S. A. Levin, T. M. Powell, and J. H. Steele, editors. Patch Dynamics. Lecture Notes in Biomathematics 96. Springer-Verlag, New York, New York, USA.

Groom M. J. and N. Schumaker. 1993. Evaluating landscape change: patterns of worldwide deforestation and local fragmentation. Pages 24-44 *in* P. M. Kareiva, J. G. Kingsolver, and R. B. Huey, editors. Biotic Interactions and Global Change. Sinauer, Sunderland, MA, USA.

Schumaker, N. H. 1995. Habitat connectivity and spotted owl population dynamics. Unpublished dissertation.

Schumaker, N. H. *In Review*. Using landscape indices to predict habitat connectivity.

Schumaker, N. H. *In Prep*. Spotted owl population trends on Washington's Olympic Peninsula.

Papers Presented At Meetings

3/92

Linking demographic models to a Geographic Information System. Pacific Ecology Conference.

8/92

Using a spatially explicit model of population viability to compare continuous-time and steady-state predictions of species persistence. Ecological Society of America - Annual Meeting.

Invited Seminars

4/94

Using landscape indices to predict habitat connectivity. University of California at Santa Cruz, Board of Environmental Studies.

10/94

Habitat fragmentation and animal movement: can valid generalizations be made? Oregon State University, Department of Forest Sciences.