How does environmental variation effect fitness of density-dependent habitat selectors?

Jody T. MacEachern

Department of Biology

Lakehead University

Submitted to Lakehead University

31 March 2010

In partial fulfillment of M.Sc. degree requirements

Committee Members:

Dr. Douglas W. Morris (Supervisor)

Department of Biology

Dr. Stephen Hecnar

Department of Biology

Dr. Brian McLaren

Faculty of Natural Resources Management

Your file Votre référence
ISBN: 978-0-494-71768-4
Our file Notre référence
ISBN: 978-0-494-71768-4

Table of Contents

List of Figures

List of Tables

List of Appendices

Abstract

The spatial distribution of animals can arise through a variety of habitat-selection strategies. It is unclear which habitat characteristics lead to the evolution of one of these strategies over another. Thus I use an individual-based model of habitat selection to assess how the mean and standard deviation of breeding-site quality in a landscape of two habitats influence the geometric mean fitness of ideal free, ideal pre-emptive and ideal despotic habitat-selection strategies. Computer simulations revealed little difference in fitness among strategies. Most simulated habitats supported large populations that saturated breeding sites and fluctuated around their carrying capacities. Despotism yielded the highest geometric mean fitness when more-or-less homogeneous sites were of low average quality. The rank order of strategies by fitness depended on density and was consistent across all simulations. Despotic habitat selectors consistently possessed the highest geometric mean fitness at low density suggesting that despotism can invade other pure strategies. The results imply that multiple habitat-selection strategies may coexist in the same population. Coexisting strategies are most likely to occur at high population density or under conditions that cause frequent variation in population size.

Keywords: evolution, habitat selection, ideal despotic, ideal free, ideal pre-emptive, individual-based model

Introduction

Most species are distributed in landscapes consisting of habitats of varying quality. Individuals choosing one habitat over another can do so by a variety of mechanisms. Alternatives include passive dispersal (McPeek and Holt 1992), as well as more complex adaptive strategies of density-dependent habitat selection (Fretwell and Lucas 1969, Morris *et al.* 2004). The success of any habitat-selection strategy will be influenced by the quality and distribution of habitats in the landscape in which individuals reside.

In the classic ideal free distribution (Fretwell and Lucas 1969), individuals are assumed to accurately assess their potential fitness among habitats and choose the habitat where their fitness is highest. An individual's fitness may, however, be constrained by the behaviour of dominants that usurp the best territories and interfere with the habitat choices of subordinates (ideal despotic distribution; Fretwell and Lucas 1969). The distribution of individuals among habitats may also be modified if individuals pre-empt use of the best sites in the landscape (ideal pre-emptive distribution; Pulliam and Danielson 1991), are constrained by the optimal choices of related individuals (Morris *et al.* 2001), or if the animals are not ideal, such as when they are unable to accurately assess fitness in a given habitat (Abrahams 1986).

Each habitat-selection strategy has been introduced in theoretical studies as a single favourable alternative to passive dispersal and occupation, often with the assumption that there is negligible cost to habitat choice (Pulliam and Danielson 1991, Rodenhouse *et al.* 1997). Individuals in a population may, however, use more than one habitat-selection strategy (Pusenius and Schmidt 2002). Ideal individuals can either choose habitats based on mean habitat quality, or they might select sites that differ in quality (Morris 2003). We do not know what habitat characteristics lead to the evolution of alternative ideal habitat-selection strategies, so it is clear that we must further explore the conditions which favour the evolution of one strategy over another. Thus, I use individual-based computer simulations to address the question: how do the mean and variance of site quality effect the evolution of habitat selection? I answer the question by contrasting three adaptive density-dependent habitat selection strategies (ideal free [IF], ideal despotic [ID], and ideal pre-emptive [IP]) and compare their fitness against a minimal-selection (MS) model, in which individuals choose the first suitable site they encounter, as a well as a fitness maximizing strategy (WMAX). I explore the dynamics of these strategies in landscapes consisting of habitats differing in mean and variance of site quality. Territorial behaviour is arguably the most extreme form of habitat selection, so I concentrate on identifying conditions under which the evolution of despotism is favoured or hindered.

Methods

Modelling habitat selection

I model the behaviour of an asexual, semelparous species using an individual-based model developed in Python 2.5.4 (Appendix V). Offspring form a common pool before selecting habitat and breeding sites. Individuals have sole use of the site they occupy. Habitat selection takes place in a model landscape consisting of 1000 breeding sites distributed equally between two habitats; population size is, therefore, a direct measure of population density. The sequence of population dynamics is recruitment followed by dispersal and mortality (Figure 1).

The quality of each breeding site is measured by the net reproductive rate that an individual achieves by occupying that site (R_o). An individual's fitness is further modified by the costs of finding, occupying and retaining the site.

Adaptive habitat-selection strategies are mimicked by varying the mechanisms (strategies) that individuals use to locate sites. The strategies differ in how many sites are sampled, and whether or not sampling is restricted to one habitat (Figure 2). All individuals, regardless of strategy, aspire to occupy a site that yields at least one descendent (aspiration level; *i.e.* Posch *et al.* 1999).

Individuals pay a search cost for every site they sample. Individuals can assess a site that is already occupied, but only ideal despotic individuals can usurp the site (see below). Costs are incorporated as deductions from the quality of the site that the individual chooses. Searching ends when costs accumulate to a threshold value (cost threshold). Upon reaching its cost threshold, the individual will occupy the best unoccupied site that it found. If no empty sites were sampled, the individual remains in the final habitat it sampled as a non-breeding individual (floater). Floaters do not occupy breeding sites, but do depress the fitness of all breeding individuals in the habitat by an equal amount. Each floater is assumed to consume enough resources to maintain itself without reproduction and thus reduces the sum of R_o achieved by breeding individuals in the habitat by 1. Floaters can arise in site-dependent strategies (Rodenhouse *et al.* 1997) whenever site seekers frequently encounter site holders (Sergio *et al.* 2009). I reasoned

Figure 1: Life cycle diagram for the asexual model species. The sequence of population dynamics is recruitment followed by dispersal and stochastic mortality. Populations are censused once each generation.

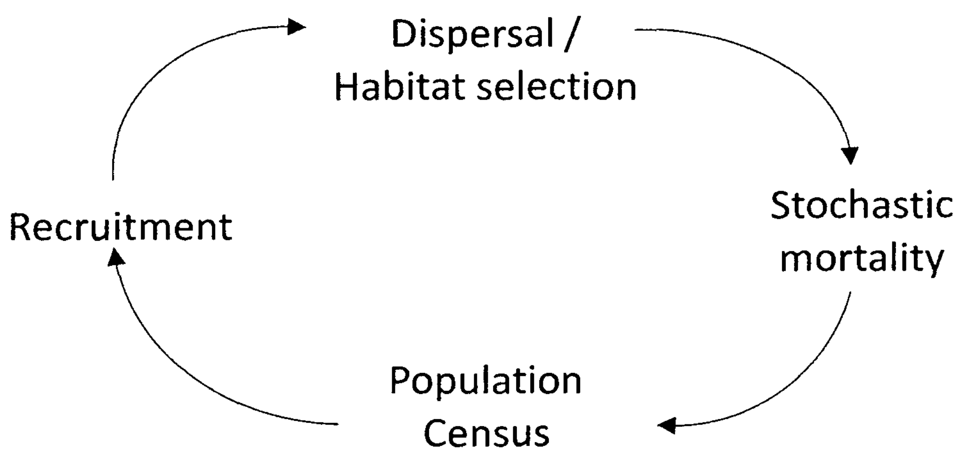
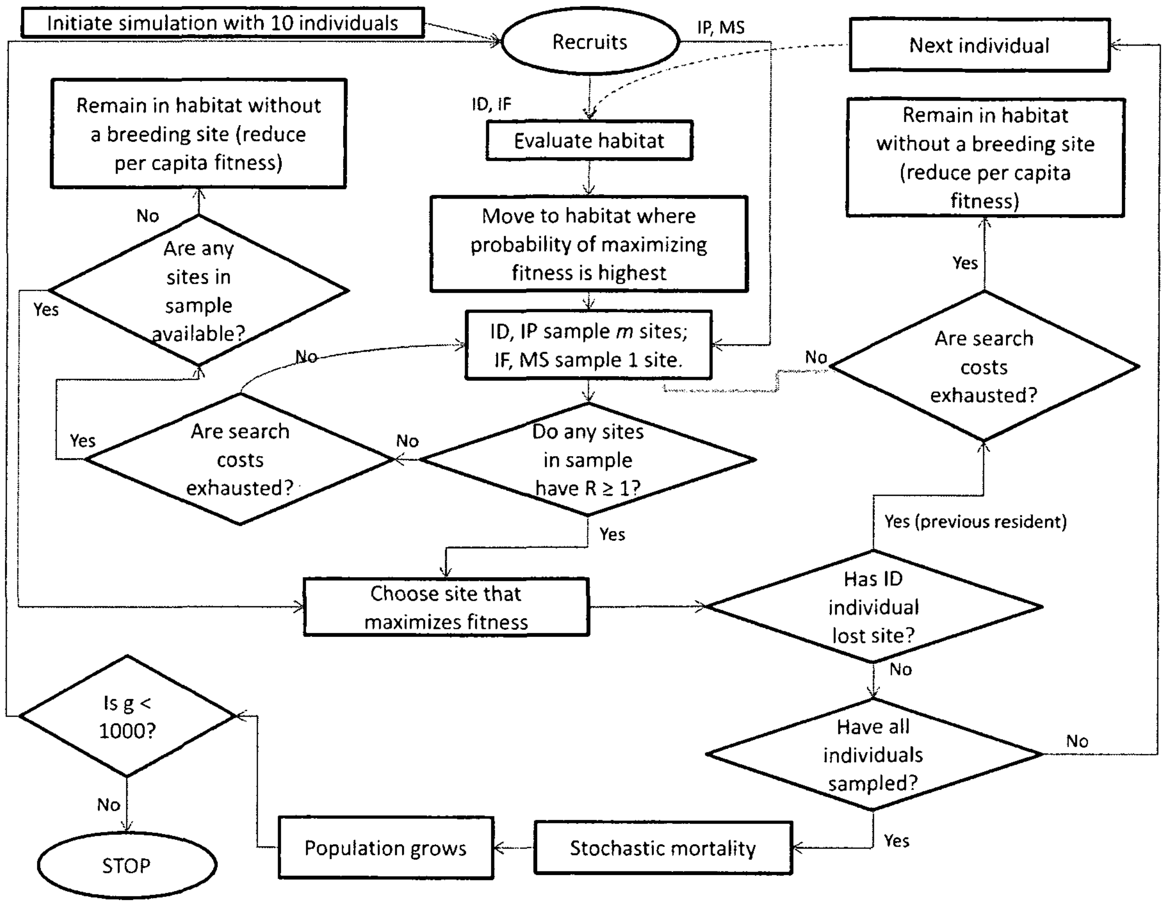

Dispersal /
Habitat selection
Stochastic
mortality
Population
Census
Recruitment

Figure 2: Flow chart of computer simulations that model four different habitat-selection strategies (g = number of generations, MS = minimal selection, IF = ideal free, ID = ideal despotic, IP = ideal pre-emptive).


Initiate simulation with 10 individuals
Recruits
IP, MS
Next individual
ID, IF
Evaluate habitat
Remain in habitat without a breeding site (reduce per capita fitness)
Move to habitat where probability of maximizing fitness is highest
Remain in habitat without a breeding site (reduce per capita fitness)
No
Are any sites in sample available?
Yes
ID, IP sample m sites; IF, MS sample 1 site.
No
Yes
Are search costs exhausted?
No
Yes
Are search costs exhausted?
No
Do any sites in sample have R ≥ 1?
Yes
Yes (previous resident)
Choose site that maximizes fitness
Has ID individual lost site?
No
Is g < 1000?
Have all individuals sampled?
No
No
STOP
Population grows
Stochastic mortality
Yes

that this will occur in populations above carrying capacity, when per capita fitness becomes negative.

Individuals using the minimal-selection strategy occupy the first site with $R_o > 1$ that they find while sampling from the entire landscape. Ideal-free individuals sample similarly except that they choose the first unoccupied site (with $R_o > 1$) from the habitat that maximizes the total value of all unoccupied sites minus the number of floaters (the habitat yielding the highest expected fitness). This simple metric implicitly incorporates the density dependence of site availability and quality, as well as the density-dependent effects of floaters. In theory (Fretwell and Lucas 1969), ideal free habitat selectors scramble for resources and have equal effects on one another's fitness. I imagined that the scramble would take place only in sites where reproduction was positive, and I modelled its effect by allowing individuals to choose the first unoccupied site encountered with a site quality greater than one. Thus, as IF or MS individuals accumulate in a habitat, the per capita fitness in the habitat should decline in direct proportion to the number of individuals living there.

Ideal despotic and ideal pre-emptive individuals sample a minimum number of sites before selecting one to occupy. If none of the sampled sites is suitable, then the individual undertakes another search until it finds a site or exceeds its cost threshold. Ideal pre-emptive individuals sample from the entire landscape whereas ideal despotic individuals sample only from the best habitat (highest expected fitness). Resident despots pay a defense cost each time their site is sampled by another individual. If an ideal despotic individual samples an occupied site in which the resident has accrued higher costs than the searcher, the searching individual can usurp that site but pays a confrontation cost to do so. The ousted individual will resume its search for a new site, retaining search costs from its previous search. If search costs from the previous search are above the cost threshold, the individual becomes a floater in the habitat it previously occupied.

The density-dependent fitness of each strategy is compared against a fitness maximizing strategy whereby individuals choose the best available site in the landscape without cost (WMAX) (Table 1).

Assessing population dynamics

The model consists of two phases, a population dynamics phase and a separate density-dependent fitness assessment phase. In the population dynamics phase, pure populations of each strategy grow in isolation from other strategies, but in identical habitats. After all individuals have chosen breeding sites (or habitats by floaters), the population suffers stochastic mortality. The frequency and severity of stochastic mortality are drawn from separate uniform distributions. The model then adds:

$$\sum (R_i - c_i) - V_T \qquad (1)$$

individuals to each habitat, where R_i is the value of a site occupied by individual i, c_i is the total cost accrued by that individual and V_T is the total number of non-breeding individuals in the habitat. Each simulation records population dynamics for 1000 generations before assessing density-dependent fitness.

Assessing density-dependent fitness

For each landscape I assess the density-dependent fitness (calculated as the geometric mean of per capita fitness – equation (1) / N, e.g., Levins 1962) of each

Table 1: A summary of model parameters and symbols.

Symbol	Description
n	Number of individuals in one habitat.
V	Number of non-breeding individuals (floaters).
R	Site value = net reproductive rate.
C	Total fitness cost = total reduction in the number of offspring produced by individuals (i.e., the sum of all costs listed below).
Model Parameters	
sc	Search cost – Reduction in the number of offspring produced for every site sampled.
dc	Defense cost – Reduction in the number of offspring produced by an ID resident when its site is sampled by another individual.
cc	Challenge cost – Reduction in the number of offspring produced by an ID individual when it usurps a site occupied by a different ID resident.
m	Sample effort – The minimum number of sites sampled by ID and IP individuals.
ct	Cost threshold – the maximum reduction in number offspring produced that individuals will accept before ceasing to search for new sites.
g	Number of generations.
sf	Stochastic frequency – The probability ($1/sf$) that a stochastic event will occur in any given year.
sv	Stochastic severity –The maximum value defining a uniform distribution with a minimum of 0, from which the percent mortality of stochastic events is drawn.

strategy at fixed population densities starting at N=10. Subsequent steps increment population density by 10, until the final step when N=1000. Each step begins by establishing a fixed population size and allowing individuals to distribute themselves among sites in the landscape according to their respective strategies. The model records the resulting distribution of individuals, site qualities and costs, then calculates the mean of values over nine replications. The level of replication was determined by balancing the need for replication with the prohibitively-long simulation time for each replicate (typically 15 hours on a PC with a 2.1GHz triple core processor and 3 GB of RAM).

Identifying winning strategies

I used two sets of simulations to search for scenarios in which the ID strategy yielded higher geometric mean fitness than either IP or IF strategies (Table 2). I first compared ID against IP strategies. I made two predictions. 1) If two habitats have the same mean and variance in site quality, then IP habitat selectors should accrue per capita fitness at least as high as ID selectors because the strategies sample from the same probability distribution. 2) If the variance in site quality is identical in the two habitats, but the mean site quality is higher in one than in the other, then the ID strategy should yield a higher geometric mean fitness than IP because ID individuals restrict sampling to the best habitat.

Accordingly, I maintained a constant variance in site quality, allowed the means to diverge between habitats, and searched for a minimum difference in mean site quality between habitats that led to the ID strategy accruing more fitness than the IP strategy. Simulations maintained a low constant mean site quality in habitat B ($\overline{X_B} = 1$) while they iteratively increased the mean site quality in habitat A (Table 2).

Table 2: A list of simulations used to assess habitat-selection strategies under 17 scenarios representing differences in the mean and variance of site qualities between two habitats. All simulations were replicated eight times. For all simulations: $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every year), $sv = 20$.

Simulation	Replicates	Mean site quality in habitat A	Standard deviation in habitat A	Mean site quality in habitat B	Standard deviation in habitat B
Differences in mean site quality	8	0.75	0.2	1	0.2
Differences in mean site quality	8	1	0.2	1	0.2
Differences in mean site quality	8	1.25	0.2	1	0.2
Differences in mean site quality	8	1.5	0.2	1	0.2
Differences in mean site quality	8	1.75	0.2	1	0.2
Differences in mean site quality	8	2	0.2	1	0.2
Differences in mean site quality	8	2.5	0.2	1	0.2
Differences in mean site quality	8	3	0.2	1	0.2
Differences in mean site quality	8	3.5	0.2	1	0.2
Differences in mean site quality	8	4	0.2	1	0.2
Differences in standard deviation	8	1	0.01	1.5	0.01
Differences in standard deviation	8	1	0.05	1.5	0.05
Differences in standard deviation	8	1	0.1	1.5	0.1
Differences in standard deviation	8	1	0.2	1.5	0.2
Differences in standard deviation	8	1	0.3	1.5	0.3
Differences in standard deviation	8	1	0.4	1.5	0.4
Differences in standard deviation	8	1	0.5	1.5	0.5

Next, I contrasted ID with IF. I predicted that IF populations should yield higher geometric mean fitness than ID populations in invariant habitats because all sites are identical. The extra cost of sampling many sites by ID reduces its per capita fitness. If, however, site qualities in one habitat are more variable than those in the other, then the ID strategy should yield higher mean fitness because ID individuals have a greater probability of finding the best sites. I initiated these simulations with a mean site quality in habitat A of 1 and a mean site quality in habitat B of 1.5. I then iteratively increased the standard deviation of site quality in each one from 0.01 to 0.5 to find scenarios in which ID yielded a higher geometric mean fitness than IF (Table 2). All simulations were replicated eight times and the grand mean of geometric mean fitness was used to rank strategies.

Finally, I conducted a preliminary sensitivity analysis to explore the relative effects of low and high values in key model parameters on fitness and population dynamics. I changed only one parameter at a time, and set remaining parameters at the values used in simulations exploring population dynamics. I explored values of the cost threshold such that, with a low cost threshold, ID and IP populations sampled, at most, only five additional sites beyond their minimal sample effort; ID and IP doubled their sample effort if no suitable sites were found under a high cost threshold, I doubled sample effort (the minimum number of sites ID and IP individuals will sample before choosing one to occupy) for the high treatment, and reduced it to one site for the low treatment. I explored challenge cost at 10^{-1} the values used in the simulations for the low treatment, and high enough (0.05) so that only the best sites would be usurped in the high treatment. I explored search costs and defense costs at values 10^{-1} and 10 times the values used in the simulations exploring population dynamics. I did not replicate the simulations because I was interested only in detecting large changes in fitness and population dynamics.

Analysis

I calculated the grand mean and standard deviation of geometric mean fitness across the eight replicates of each simulation corresponding to a different site-quality scenario. I inferred that a particular habitat-selection strategy should evolve in scenarios where it had the highest geometric mean fitness [any difference in fitness, no matter how small, is evolutionarily significant (Fisher 1930)].

In order to test my *a priori* predictions, I plotted geometric mean fitness against the mean or standard deviation of site quality. Population size is arguably a better surrogate of fitness in stochastic environments than is growth rate (Benton and Grant 2000). Thus, when comparing two strategies with approximately equal geometric mean fitness, I inferred that the strategy maintaining a higher mean population size had higher fitness. I used this "fitness assessment rule" whenever populations were maintained near their carrying capacities (because populations that maintain density near carrying capacity all possess a per capita fitness (R_o) near one).

I reasoned that differences in geometric mean fitness among strategies might arise through differences in status (breeders vs. floaters) and in the distribution of individuals between habitats. I tested for differences in distribution (number of individuals occupying the two habitats) among strategies with two general log-linear analyses (SPSS v18): one analyzing data from a simulation in which the ID strategy ranked first in fitness, and appeared to have a different distribution of individuals compared with other strategies, and one where ID ranked last in fitness.

I refined my search for "winning strategies" by using the fitness assessment data to map each strategy's fitness across an adaptive landscape for habitat selection (Wright 1932). I mapped geometric mean fitness against density of each simulation for each habitat-selection strategy (Morris 2003). Different strategies "win" at different densities, so I assessed the growth rate (fitness) at low density as an indicator of a mutant's ability to invade other pure strategies (Mylius and Diekmann 1995). In order to assess the success of mutant strategies under different scenarios, I plotted geometric mean fitness at low density against the mean or standard deviation (depending on which varied in the simulation) in site quality. I used data from only the first three generations because populations typically grew to carrying capacity within four generations.

Results

The minimal-selection strategy was more susceptible to extinction than other strategies

The minimal selection strategy often went extinct when mean site quality was low ($\bar{X}_B$= 1; $\bar{X}_A$= 0.75 or 1; 4 extinctions in 16 replicates). These frequent extinctions lowered the grand mean of geometric mean fitness among replicates (Figure 3). All other strategies attained, and maintained, high mean fitness.

All strategies accrued similar fitness and population size

There was little difference in the geometric mean fitness among strategies (Figures 3, 4, 5). The similarity in geometric mean fitness among strategies occurred because populations saturated both habitats quickly, and then fluctuated little around their

Figure 3: The geometric mean fitness of four different simulated habitat-selection strategies when mean site quality in habitat A is low. IF, ID and IP populations accrued similar fitness. MS had lower and more variable mean fitness because it became extinct in 4 of 16 simulations (arrows). Error bars represent one standard deviation. For all simulations: $\overline{X_b} = 1.5$, $\sigma_a = 0.2$, $\sigma_b = 0.2$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every generation) and $sv = 20$.

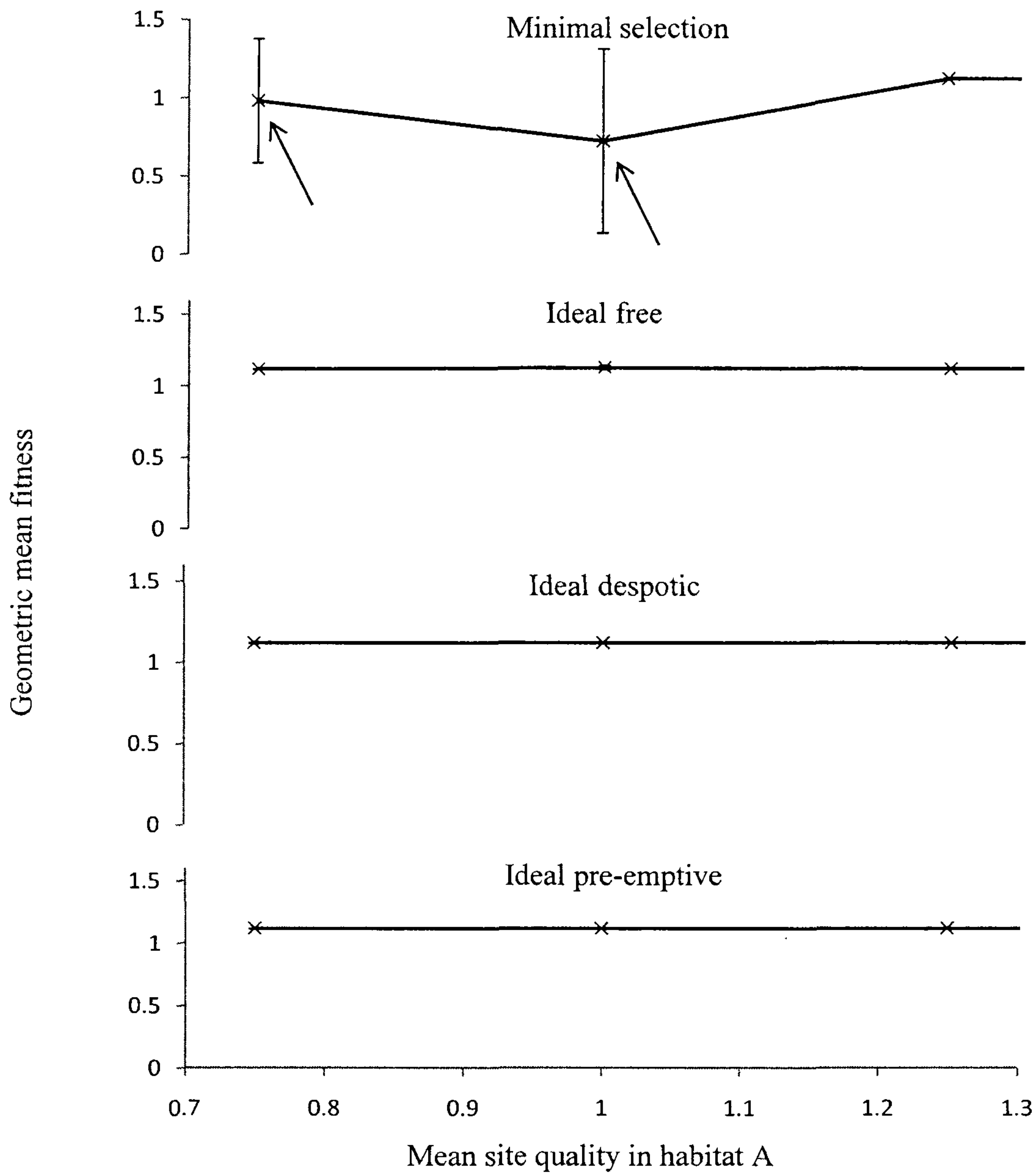
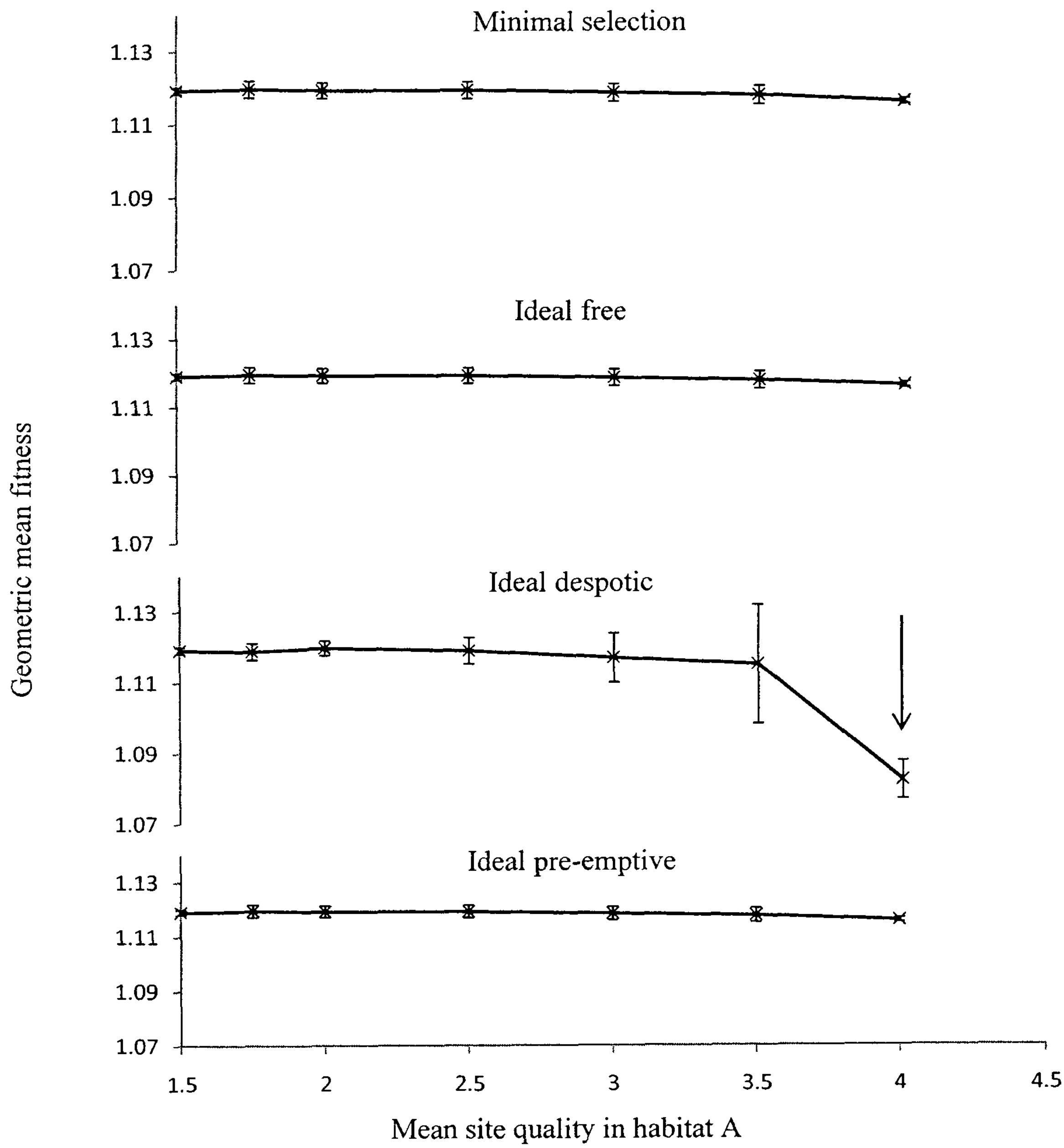
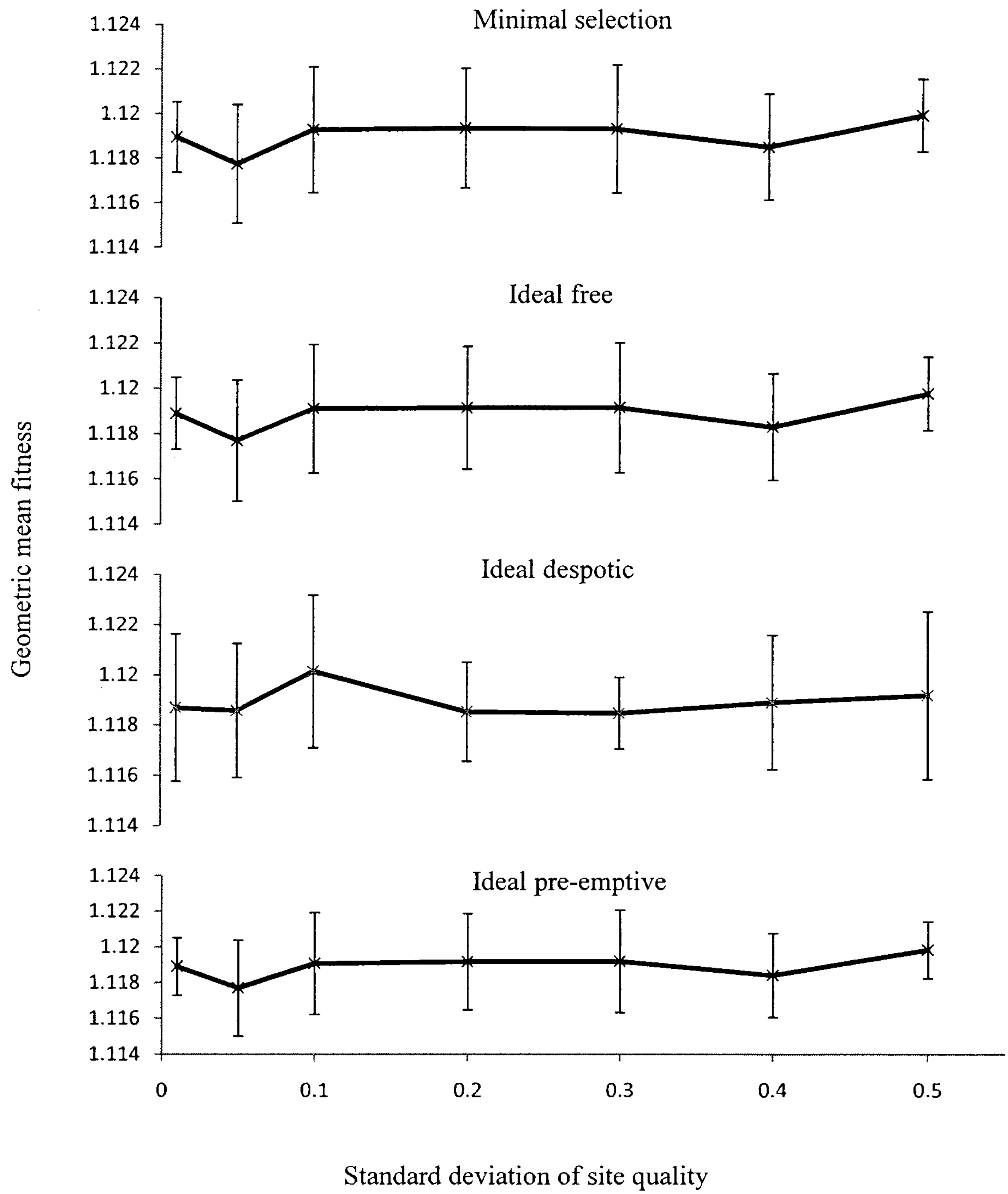

Minimal selection
Ideal free
Ideal despotic
Ideal pre-emptive
Geometric mean fitness
Mean site quality in habitat A
1.5
1
0.5
0
0.7
0.8
0.9
1
1.1
1.2
1.3

Figure 4: The geometric mean fitness of four different simulated habitat-selection strategies across a range of mean site qualities in habitat A. As mean site quality in habitat A increases, the geometric mean fitness of ID populations is more variable, and eventually lower, than other strategies (arrow) that accrued very similar fitness. Error bars represent one standard deviation. Fitness is the grand mean of 8 replications of each simulation. For all simulations: $\overline{x_b} = 1.5$, $\sigma_a = 0.2$, $\sigma_b = 0.2$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every year) and $sv = 20$.



Minimal selection
Ideal free
Ideal despotic
Ideal pre-emptive
1.13
1.11
1.09
1.07
1.5
2
2.5
3
3.5
4
4.5
Geometric mean fitness
Mean site quality in habitat A

Figure 5: The geometric mean fitness of four simulated habitat-selection strategies across a range of standard deviations in site quality. Geometric mean fitness varied more among replicates of a simulation than among different scenarios of site quality. Error bars represent one standard deviation. Fitness is the grand mean of 8 replications for each simulation. For all simulations: $\overline{x_a} = 1.5$, $\overline{x_b} = 1$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every year) and $sv = 20$.



Geometric mean fitness
Minimal selection
Ideal free
Ideal despotic
Ideal pre-emptive
1.124
1.122
1.12
1.118
1.116
1.114
0
0.1
0.2
0.3
0.4
0.5
Standard deviation of site quality

carrying capacities (Figure 6). Furthermore, there was little difference in mean population size among strategies within each simulation (Figure 7, Figure 8). Mean population growth rate in each 1000-generation simulation was thus dominated by low variance about stochastically-fluctuating densities near K.

ID populations accrued lower fitness in rich habitats than did other strategies

The ID strategy achieved similar geometric fitness as alternative strategies except when one habitat had a much higher mean site quality than the other (Figure 4). Unlike other strategies, the variance in mean fitness also increased for ID populations as habitats diverged in mean site quality. Nonetheless, ID population size was comparable with that produced by other strategies (Figure 7).

ID populations accrued the highest geometric mean fitness in only a narrow set of conditions

ID populations accrued the highest geometric mean fitness among strategies only in scenarios with low mean site quality and low variance (Figure 5). Within simulations, however, ID populations accrued the highest geometric mean fitness among strategies in only 6 of 8 simulations. In the remaining two simulations, ID had the lowest geometric mean fitness among strategies.

ID populations were distributed differently than the populations of other strategies

In both scenarios in which I analyzed the distribution of individuals there were more breeders than floaters (Table 3, low-quality habitat A: breeder, $Z = 13.2$, $P < 0.001$; high-quality habitat A: breeder, $Z = 6.1$, $P < 0.001$). Even so, ID population size was lower than other strategies when both habitats had low mean site quality (ID significant

Figure 6: Time series of population sizes produced from simulations of four different simulated habitat-selection strategies. Each population grew quickly to carrying capacity then fluctuated asynchronously in response to stochastic mortality. Parameter values as follows: $\overline{X_a}$ = 2, $\overline{X_b}$ = 1, σ_a = 0.2, σ_b = 0.2, *sc* = 0.001, *dc* = 0.0001, *cc* = 0.01, *ct* = 0.25, *m* = 10, *g* = 1000, *sf* = 1 (one stochastic event occurs every generation) and *sv* = 20.

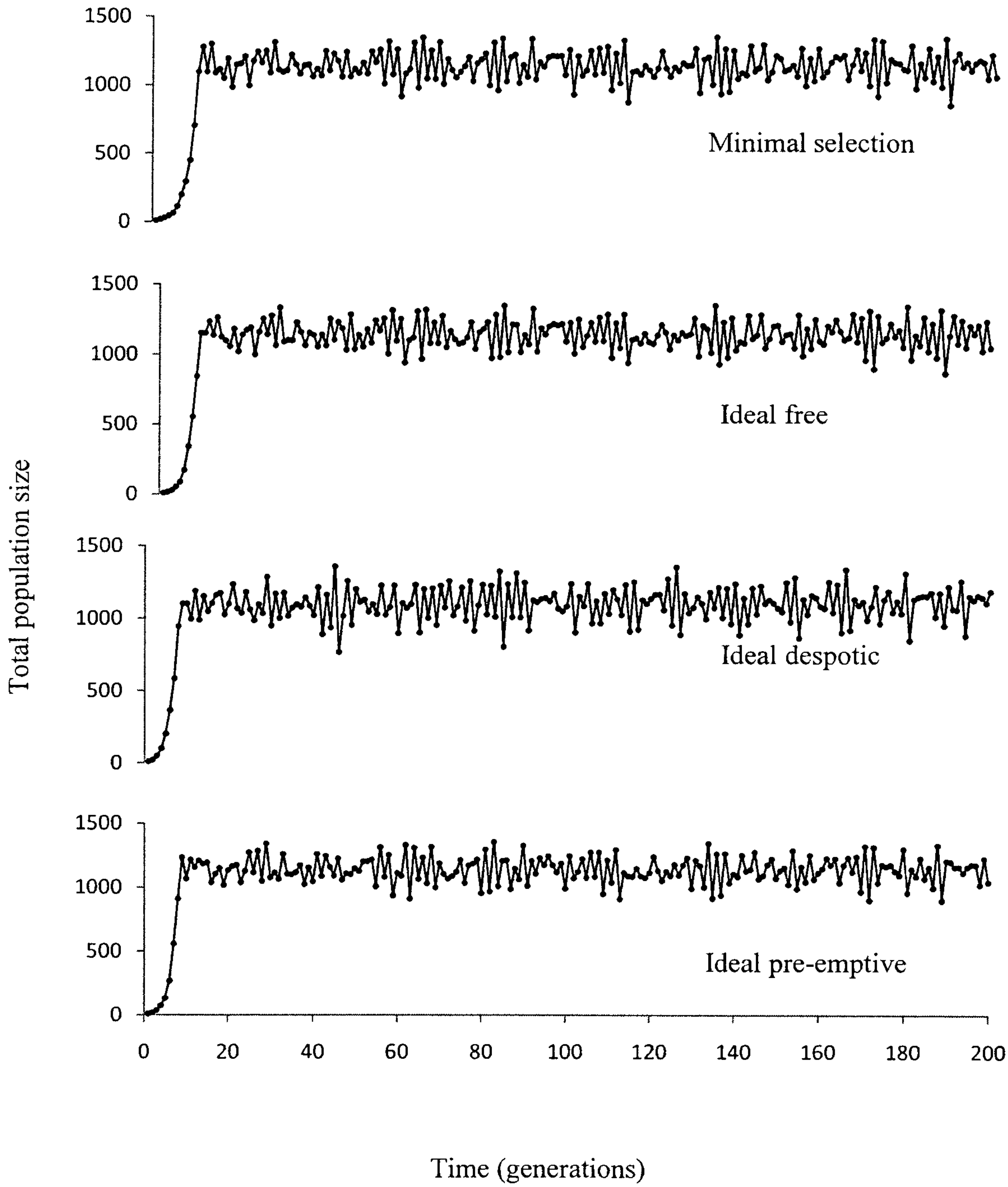
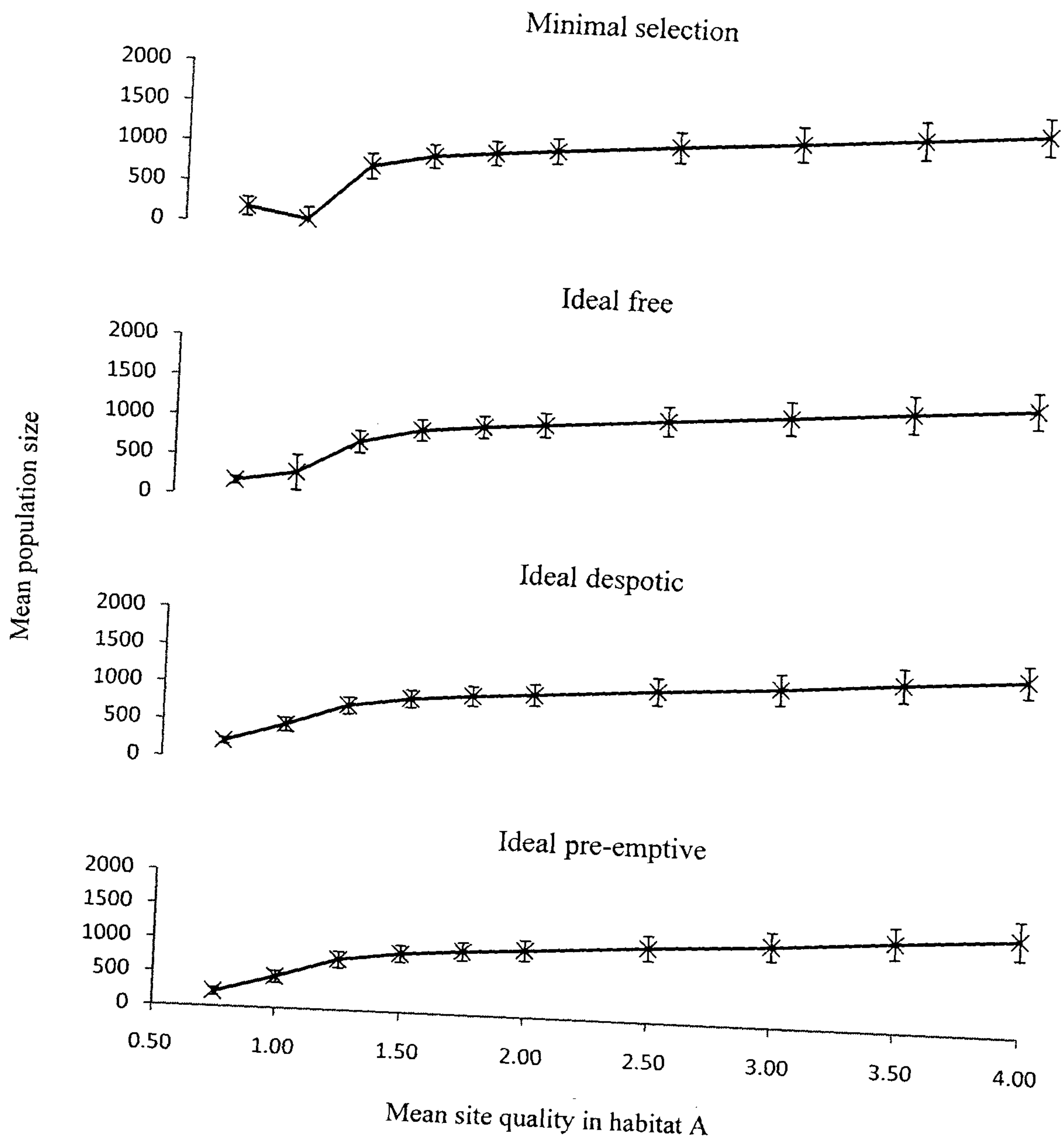

1500
1000
500
0
Minimal selection
Ideal free
Ideal despotic
Ideal pre-emptive
Total population size
0
20
40
60
80
100
120
140
160
180
200
Time (generations)

Figure 7: The mean population size of four different simulated habitat-selection strategies across a range of mean site qualities in habitat A. The mean population for each strategy is similar in most scenarios. Mean population sizes are the grand means of 8 replications of each simulation. Error bars represent one standard deviation. For all simulations: $\overline{X_b} = 1$, $\sigma_a = 0.2$, $\sigma_b = 0.2$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every year) and $sv = 20$.



Minimal selection
Ideal free
Ideal despotic
Ideal pre-emptive
2000
1500
1000
500
0
Mean population size
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
Mean site quality in habitat A

Figure 8: The mean population size of four different simulated habitat-selection strategies across a range of standard deviations in site quality in two habitats. There was little difference in mean population size among strategies in all simulations. Error bars represent one standard deviation. Mean population sizes are the grand mean of 8 replicates of each simulation. For all simulations: $\overline{X_a}$ = 1.5, $\overline{X_b}$ = 1, *sc* = 0.001, *dc* = 0.0001, *cc* = 0.01, *ct* = 0.25, *m* = 10, *g* = 1000, *sf* = 1 (stochastic events occur every year) and *sv* = 20.

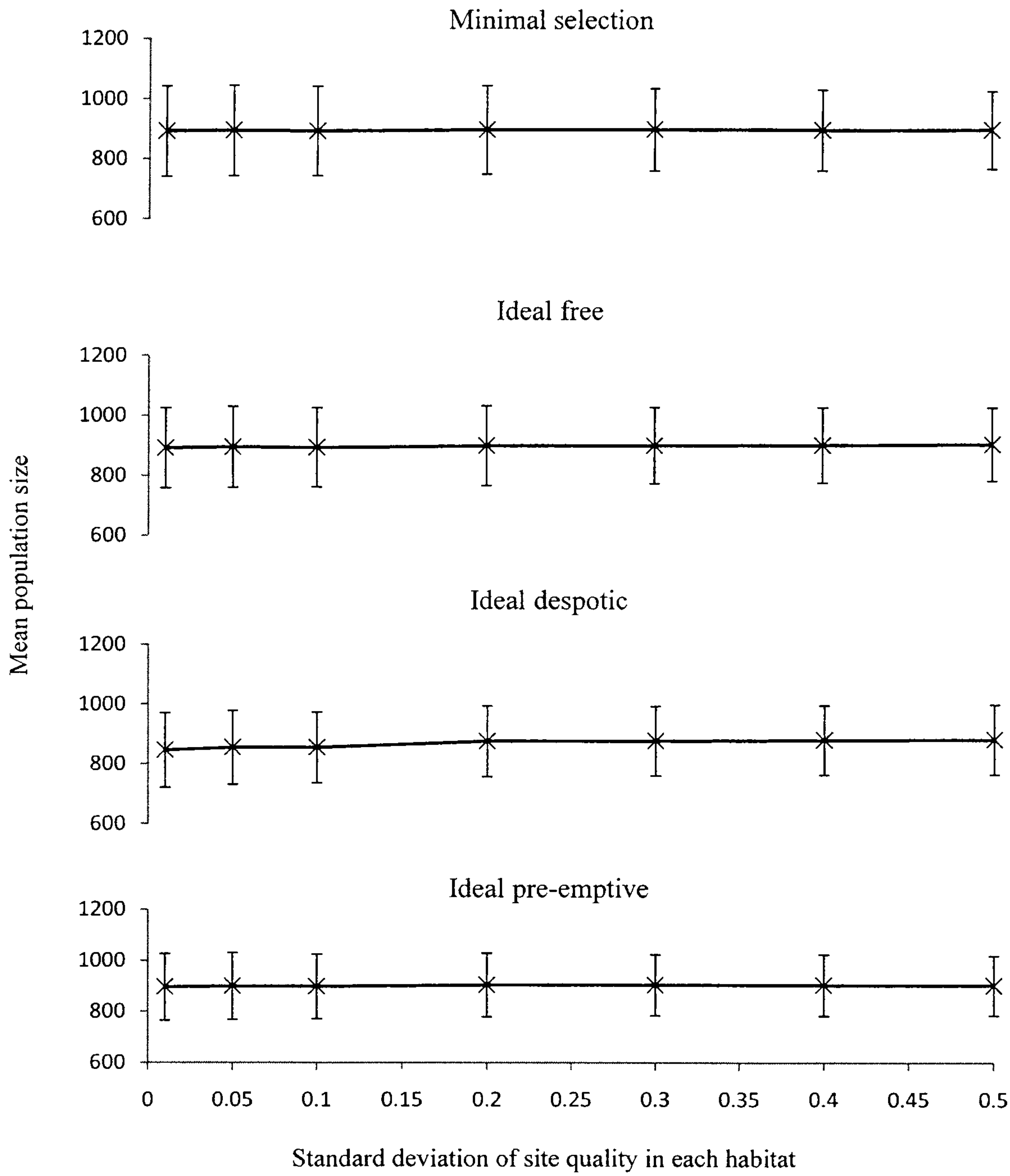

Minimal selection
Ideal free
Ideal despotic
Ideal pre-emptive
Mean population size
1200
1000
800
600
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Standard deviation of site quality in each habitat

Table 3: Results from two log-linear analyses used to compare the distribution of individuals by strategy, status (breeders/floaters) and habitat. Each scenario (analysis) includes data from a single simulation with no replication. For the low-quality habitat A scenario: $\overline{X_a} = 4$, $\overline{X_b} = 1$, $\sigma_a = 0.2$, $\sigma_b = 0.2$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ For the high-quality habitat A scenario: $\overline{X_a} = 1$, $\overline{X_b} = 1.5$, $\sigma_a = 0.1$, $\sigma_b = 0.1$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ Significant parameters are indicated in bold. Z = standardized parameters (estimate / standard error). Redundant parameters (the last parameter in each class is fully explained by the remaining parameters in that class) are not shown.

Scenario	Parameter	Estimate	Standard error	Z	Significance
Low-quality habitat A	**Constant**	2.804	0.246	11.396	<0.001
	MS	0.007	0.347	0.019	0.985
	IF	-0.084	0.356	-0.235	0.814
	ID	-1.011	0.476	-2.123	0.034
	Breeder	3.297	0.251	13.156	<0.001
	Habitat A	-0.015	0.349	-0.042	0.967
	MS * Breeder	-0.016	0.354	-0.046	0.963
	IF * Breeder	0.057	0.362	0.156	0.876
	ID * Breeder	0.971	0.481	2.019	0.044
	MS * Habitat A	0.016	0.492	0.033	0.973
	IF * Habitat A	0.013	0.504	0.026	0.979
	ID * Habitat A	1.704	0.565	3.017	0.003
	Breeder * Habitat A	-0.048	0.356	-0.135	0.893
	MS * Breeder * Habitat A	-0.011	0.501	-0.022	0.982
	IF * Breeder * Habitat A	0.031	0.513	0.061	0.951
	ID * Breeder * Habitat A	-1.743	0.573	-3.042	0.002
High-quality habitat A	**Constant**	5.643	0.060	94.809	<0.001
	MS	0.005	0.084	0.057	0.955
	IF	0.002	0.084	0.029	0.977
	ID	0.061	0.083	0.730	0.465
	Breeder	0.464	0.076	6.104	<0.001
	Habitat A	-0.001	0.084	-0.010	0.992
	MS * Breeder	-0.005	0.107	-0.050	0.960
	IF * Breeder	-0.002	0.107	-0.022	0.982
	ID * Breeder	-0.150	0.107	-1.395	0.163
	MS * Habitat A	-0.004	0.119	-0.030	0.976
	IF * Habitat A	0.002	0.119	0.018	0.986
	ID * Habitat A	-0.098	0.119	-0.822	0.411
	Breeder * Habitat A	0.002	0.107	0.016	0.988
	MS * Breeder * Habitat A	0.003	0.152	0.022	0.983
	IF * Breeder * Habitat A	-0.002	0.152	-0.015	0.988
	ID * Breeder * Habitat A	0.186	0.152	1.222	0.222

Z =-2.1, P = 0.03). Specifically, ID had a lower population size in the low-quality habitat (ID × Habitat A, Z =3.0, P = 0.003) where ID populations generally accrued higher geometric mean fitness than other strategies. Most significantly, ID populations were distinguished by disproportionately more floaters in the lower-quality habitat in comparison with other strategies (Figure 9; Table 3, ID × Habitat A × Breeder, Z =-3.0, P =0.002).

The rank order of habitat-selection strategies varied with population size

The WMAX strategy, as expected, outperformed all others (Figure 10). Ideal despotic populations accrued higher fitness than the remaining three strategies at low density, but lost second-best ranking to ideal pre-emptive habitat selectors at intermediate densities. The rankings of fitness for all strategies were consistent across simulations (not illustrated). Importantly, ideal despotic strategies consistently accrued the highest geometric mean fitness during population growth at low density (Figure 11).

Minimal-selection strategies fail when search costs are high

The sensitivity analysis revealed high extinction probabilities for MS populations (very low fitness values in Table 4). Extinction occurred in 5 of the 10 simulations used in the sensitivity analysis: when challenge cost was low and high, when search cost was high, and in simulations where sample effort or defense cost was low. These extinctions occurred even though defense and challenge costs, and sample effort do not affect MS individuals, but MS populations grew in one simulation and went extinct in the other.

Figure 9: The abundance of breeders and floaters using four different simulated habitat-selection strategies to occupy two habitats differing in site quality. Distribution data are from two separate simulations with no replication. A) Mean site quality is lower in habitat A than in habitat B. $\overline{X_a} = 1, \overline{X_b} = 1.5$. B) Mean site quality much higher in habitat A than in habitat B. $\overline{X_a} = 4, \overline{X_b} = 1.5$. The proportion of floaters was similar for all strategies except ID (that produced a higher proportion of floaters in the poor-quality habitat (A, Table 3). For all strategies: $\sigma_a = 0.1$, $\sigma_b = 0.1$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every generation) and $sv = 20$.

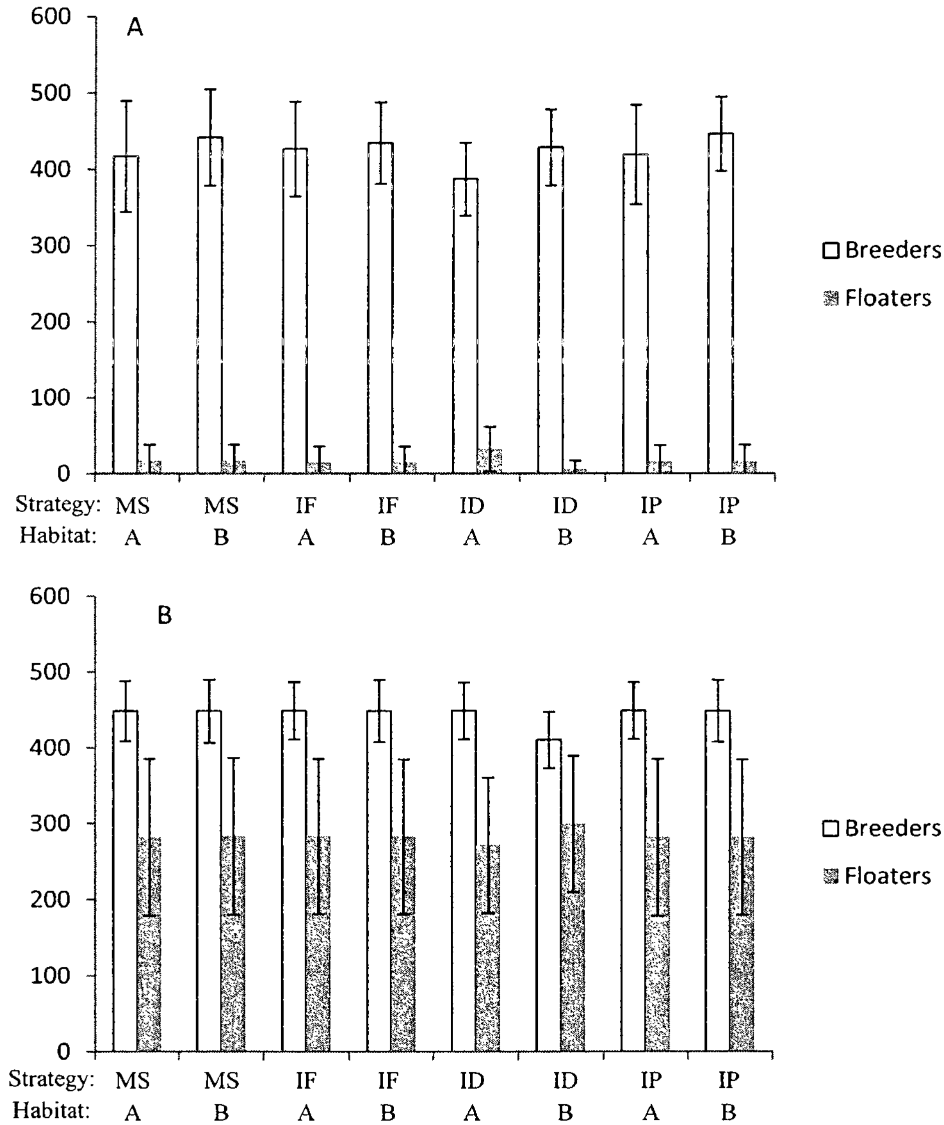
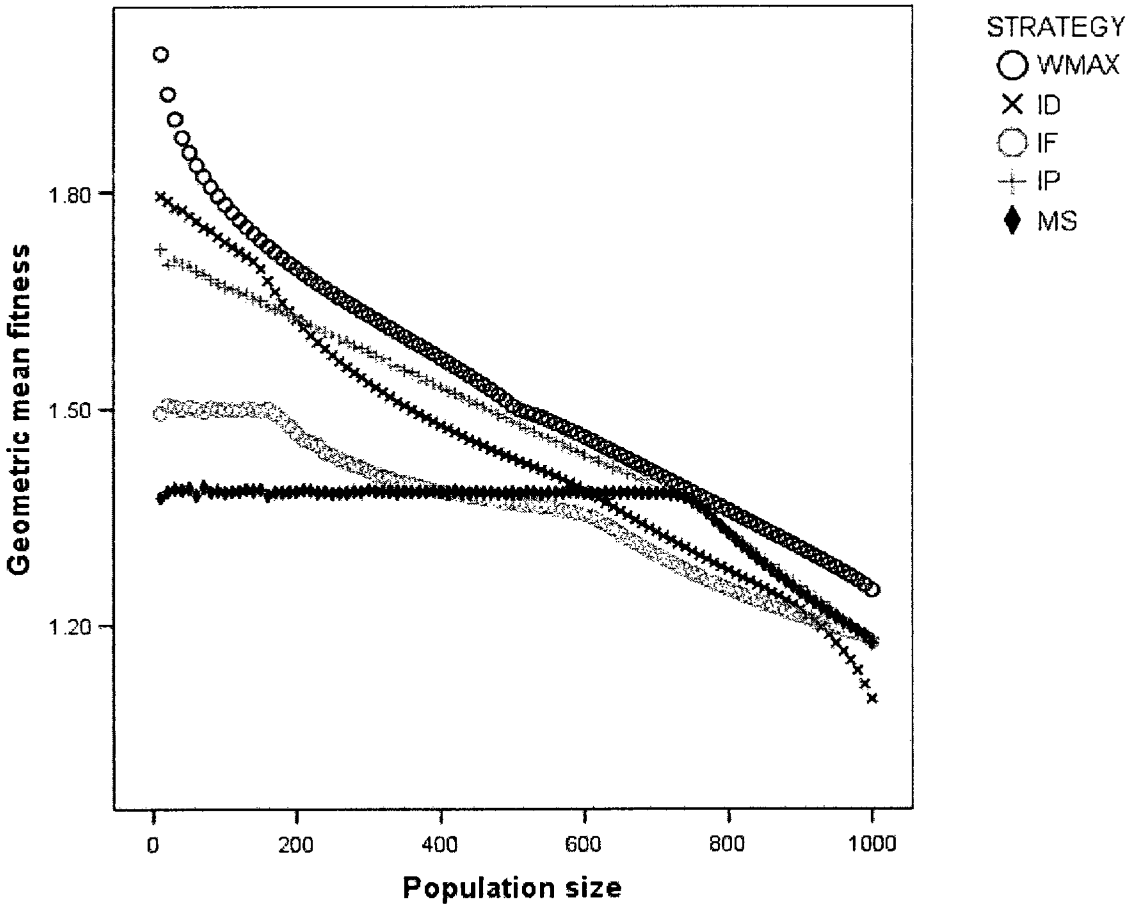

A
B
600
500
400
300
200
100
0
Breeders
Floaters
Strategy: MS MS IF IF ID ID IP IP
Habitat: A B A B A B A B

Figure 10: Fitness comparison of four different habitat selection strategies relative to the best attainable strategy (WMAX) across a range of population sizes. The ID strategy yielded the second highest geometric mean fitness at low density, and the lowest mean fitness at high density. For all strategies: $\overline{x_a} = 1$, $\overline{x_b} = 1.5$, $\sigma_a = 0.1$, $\sigma_b = 0.1$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every generation) and $sv = 20$.



STRATEGY
WMAX
ID
IF
IP
MS
1.80
1.50
1.20
Geometric mean fitness
0
200
400
600
800
1000
Population size

Figure 11: The geometric mean fitness from simulations of four different habitat-selection strategies during the first three generations of population growth. A) Geometric mean fitness across simulations that varied the mean site quality in habitat A ($\overline{x_b}$ = 1, σ_a = 0.1, σ_b = 0.1). B) Geometric mean fitness of the four strategies across simulations that varied the standard deviation of site quality. The ID strategy consistently yielded the highest mean fitness during early population growth. $\overline{x_a}$ = 1, $\overline{x_b}$ = 1.5. Geometric mean fitness values are the grand mean of 8 replicates of each simulation. For all simulations: *sc* = 0.001, *dc* = 0.0001, *cc* = 0.01, *ct* = 0.25, *m* = 10, *g* = 1000, *sf* = 1 (stochastic events occur every generation) and *sv* = 20.

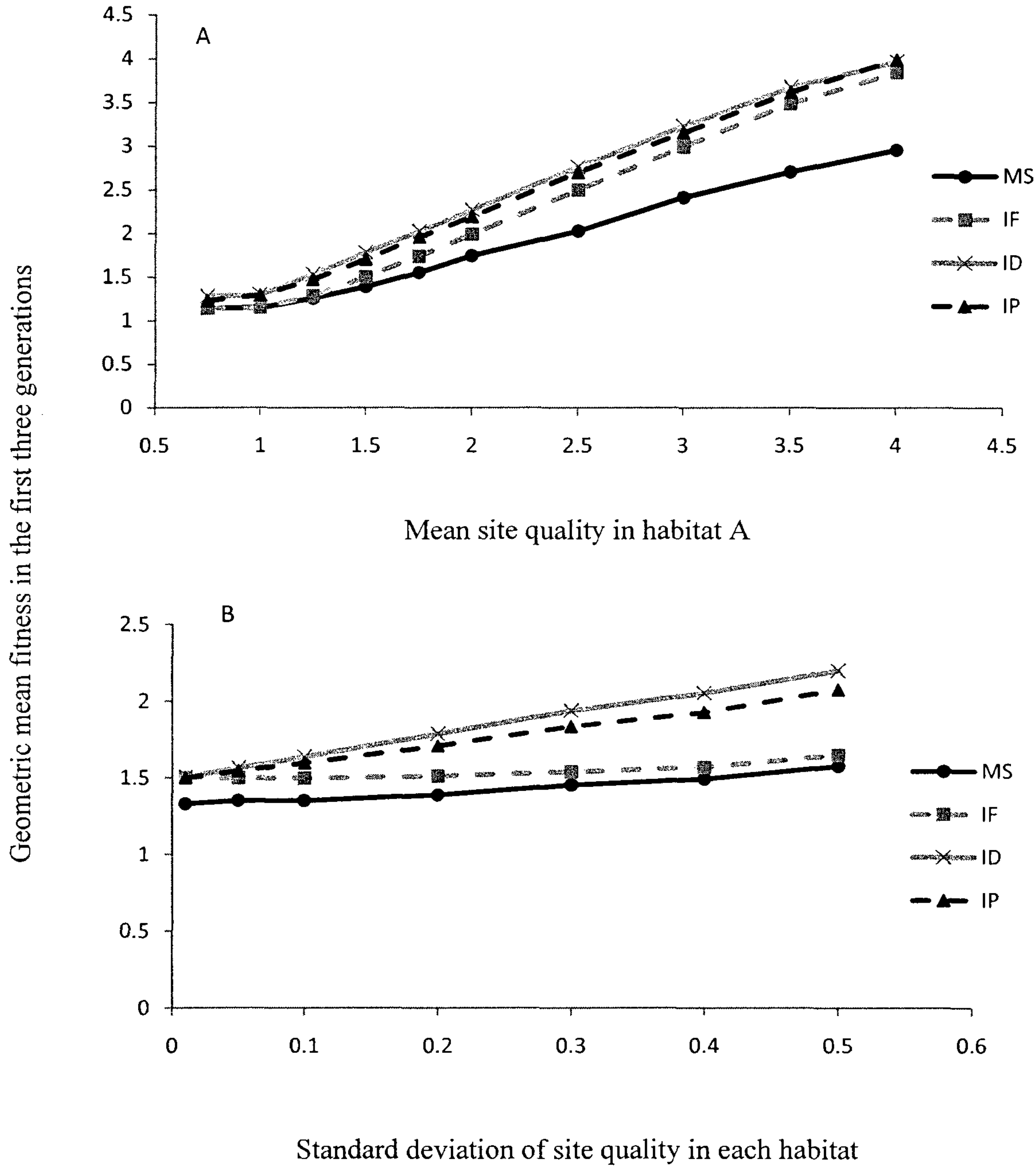

Geometric mean fitness in the first three generations
A
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
MS
IF
ID
IP
Mean site quality in habitat A
B
2.5
2
1.5
1
0.5
0
0
0.1
0.2
0.3
0.4
0.5
0.6
MS
IF
ID
IP
Standard deviation of site quality in each habitat

Table 4: Geometric mean fitness of four simulated habitat-selection strategies when sampling effort and costs are high or low. Each scenario represents a single simulation with no replication.

							Geometric Mean Fitness			
Variable Assessed	Treatment	Search Cost	Defense Cost	Challenge Cost	Cost Threshold	Sample Effort	MS	IF	ID	IP
Challenge Cost	Low	0.001	0.0001	0.001	0.25	10	0.001	1.136	1.118	1.118
Challenge Cost	High	0.001	0.0001	0.05	0.25	10	0.083	1.132	1.122	1.122
Search Cost	Low	0.0001	0.0001	0.01	0.25	10	1.156	1.140	1.118	1.118
Search Cost	High	0.01	0.0001	0.01	0.25	10	0.000	1.138	1.124	1.124
Sample Effort	Low	0.001	0.0001	0.01	0.25	1	0.003	1.121	1.121	0.016
Sample Effort	High	0.001	0.0001	0.01	0.25	20	1.164	1.157	1.118	1.118
Cost Threshold	Low	0.001	0.0001	0.01	0.15	10	0.602	1.145	1.118	1.118
Cost Threshold	High	0.001	0.0001	0.01	0.5	10	0.888	1.145	1.120	1.120
Defense Cost	Low	0.001	1.00E-05	0.01	0.25	10	0.001	1.152	1.124	1.124
Defense Cost	High	0.001	0.001	0.01	0.25	10	1.117	1.118	1.116	1.116

Costs of habitat selection had little effect on IF, ID, and IP strategies, but IP populations accrued low fitness with low sampling effort

Changing defense cost, challenge cost, search cost and cost threshold had little effect on IF, ID, and IP strategies. IP populations, however, suffered from drastically reduced geometric mean fitness when sample effort was low (Table 4).

Discussion

I evaluated the relative success of four different habitat-selection strategies by simulating how individuals found and maintained breeding sites in two adjacent habitats differing in site quality. Despite my attempts to create conditions in which single strategies would accrue conspicuously higher geometric mean fitness than alternatives, most simulations yielded similar fitness for all strategies. Regardless of these similarities, it is nevertheless clear that the ID strategy accrued higher geometric mean fitness than all other realistic strategies at low density. Although it is thus possible to imagine that the ID strategy can displace others at low density, it quickly loses its advantage as populations grow toward carrying capacity.

Multiple habitat-selection strategies might coexist in populations near carrying capacity

The similarities in geometric mean fitness and population sizes of strategies suggest the intriguing possibility that several habitat-selection strategies may coexist in stable populations near their carrying capacities. This remarkable insight contravenes the long-held intuition that competitive neglect (Ripley 1959; Hutchinson and MacArthur 1959) and resource defense (Brown 1964) cause individuals to abandon despotic behaviours at high population density. Dominant territorial individuals are known to abandon interference competition and join con-specifics in scramble competition (Myers *et al.* 1979), or abandon their territories altogether and disperse to lower-density areas (Steneck 2006). How, then, can I account for the potential coexistence of multiple strategies emerging through my simulations of habitat selection?

The key to understanding the coexistence of multiple habitat-selection strategies likely lies in underappreciated differences between dominant and subordinate individuals. Current models of despotic and pre-emptive habitat selection assume that all individuals use the same habitat-selection strategy. Strategies might coexist, however, if dominant individuals restrict interference competition to the best-quality sites, thereby allowing subordinate individuals to subsist unobtruded in poorer-quality sites. Höjesjö *et al.* (2004) report a possible example from experiments assessing growth and survival of newly-emerged brown trout in simple and complex habitats. Dominant individuals grew more quickly than subordinate fry in simple environments (sand substrate), but lost their growth advantage in complex ones (gravel and stone substrate). The costs of aggression and resource defense likely increase when subordinates retreat into complex habitats, and thus change the cost/benefit ratio of dominant behaviour. Other factors, such as dominance hierarchies, can reduce the frequency or severity of competitive interactions (Maynard-Smith and Parker 1976; Eshel and Sansone 1995), thereby promoting the coexistence of dominant and subordinate individuals. A dominance hierarchy relating to habitat selection can similarly develop through differences in growth rates (Hakoyama and Iguchi 2001).

Perhaps the best evidence of coexisting strategies comes from habitat-selection by meadow voles (*Microtus pennsylvanicus*). For example, ideal despotic and ideal free habitat selectors have been observed in a single population (Pusenuis and Schmidt 2002). Dominant individuals used undisturbed patches in an ideal despotic manner, while subordinates followed an ideal free distribution among disturbed patches. The apparent co-existence of two pure strategies is particularly intriguing because the 100-800 voles/ha observed by Pusenius and

Schmidt (2002) densities are near, or in excess of, typical meadow vole carrying capacity [population growth of meadow voles ceased at population densities ranging from 100-600 voles/ha in grassland habitats in Indiana (Lin and Batzli 2001)].

Lin and Batzli (2004) also categorized meadow voles as ideal-free habitat selectors whereas Oatway and Morris (2007) referred to them as vague density-dependent habitat selectors. Individuals living at low density may be incapable of assessing habitat differences in habitats with high carrying capacities. Hence it is clear that habitat selection is not a fixed behavioural trait, but rather emerges from plastic responses shaped by tradeoffs such as the costs and benefits of aggression versus placid behaviours.

Multiple habitat selection strategies might also coexist in populations with fluctuating density

Recall that the rank order of strategies varied with density. A pure ID strategy might thereby predominate when density is maintained well below carrying capacity (e.g., by generalist predators; Hanski *et al.* 1991). If, however, populations fluctuate (e.g., environmental stochasticity (Getz *et al.* 2006), then multiple strategies can be maintained through cyclical selection (Rosenzweig 1991). The potential of maintaining multiple strategies in stochastically varying environments is particularly important because temporal stochasticity influences our ability to detect habitat selection (Jonzén *et al.* 2001). My simulations suggest that stochasticity not only influences habitat-selection strategies, (Jonzén *et al.* 2004), but that it may also promote their coexistence. Temporal variation in habitat quality has been shown to lead to the coexistence of competing species (Schmidt *et al.* 2000), and also dominant and subordinate individuals (Höjesjö 2004).

Experiments assessing habitat selection may be biased if they ignore density-dependent habitat-selection strategies

Although it is well acknowledged that habitat selection is both density and frequency dependent (Rosenzweig 1981; Rosenzweig 1991; Morris 2003), my simulations suggest that the strategy itself also depends on density. If true, then dire consequences await those who attempt to evaluate habitat selection using fixed-density experiments (or field observations). Despotism yields higher fitness at low population density, but gives way to ideal pre-emptive habitat selection at higher densities. And, if populations are allowed to grow to carrying capacity, then all strategies may be able to coexist.

Strategies of habitat selection should be explored with an invasion analysis

An important caveat to drawing conclusions based on the geometric mean fitness of pure populations is that the geometric mean fitness of a strategy might change when multiple habitat-selection strategies coexist in the same simulation. Future simulations should explore coexistence with an invasion analysis similar to that used by Ranta and Kaitala (1999), where a rare mutant's ability to invade a pure population is assessed simultaneously with a pure population's resistance to invasion. It may be necessary, however, to contrast all possible combinations of strategies in order to assess their potential for invasion, resistance, and coexistence. The invasion analysis will also be complicated if individuals use their behavioural flexibility to play mixed strategies of habitat choice. Regardless of these complexities, despotic habitat selection should have an invasion advantage over other pure strategies owing to its high growth rate at low density (e.g., Mylius and Diekmann 1995). This result is intuitively satisfying because, in my simulations, only ideal-despotic individuals could oust individuals using other habitat-selection strategies. Successful invasion, however, will depend on population dynamics and its interaction with habitat quality. Although ID populations at low density possess higher mean fitness, poor-quality sites will be unoccupied and allow for coexistence of other strategies. The cost of despotic behaviours will increase with increasing population density, and may provide an opportunity for replacement by other strategies. These possibilities should be especially intriguing for those ecologists who believe that territorial behaviour stabilizes population dynamics. The simulations completed here suggest that despotic habitat selection may persist only through "re-invasion" in highly fluctuating populations.

Habitat selection is often viewed as a fixed trait of a species. My simulations suggest that habitat-selection strategies are not fixed traits, and that coexisting strategies are not only possible, but likely. This interpretation has important implications for assessing habitat-selection strategies. The possibility of multiple coexisting habitat selection strategies complicates our ability to assess density-dependent habitat selection in populations; however, it also opens a new, largely unexplored and exciting avenue for our understanding of how animals use and distribute themselves among habitats.

Acknowledgements

I sincerely thank E. Sauks, my steadfast supporter and motivator throughout this experience. Sincerest thanks also to D. Morris for his unending patience and support, and a wonderful experience throughout the last several years. Thanks to D. and K. Morris also for the hospitality. Thanks to the Morris lab for all the discussions, and reviews and comments on the manuscript. Special thanks to V. Sundararaj and J. Levac for more thorough reviews of earlier drafts, as well as Drs. S. Hecnar, B. McLaren and P. McLoughlin. Finally, I'd like to thank Canada's Natural Sciences and Engineering Research Council for its continuing support of DWM's research in evolutionary ecology.

Literature Cited

Abrahams, M.V. 1986. Patch choice under perceptual constraints: a cause for departures from an ideal free distribution. Behav. Ecol. Sociobio. 19: 409-415.

Benton, T.G. and Grant A. 2000. Evolutionary fitness in ecology: Comparing measures of fitness in stochastic, density-dependent environments. Evol. Ecol. Res. 2: 769-789.

Brown, J.L. 1964. The evolution of diversity in avian territorial systems. Wilson Bull. 76: 160-169.

Eshel, I. and Sansone, E. 1995. Owner-intruder conflict, Grafen effect, and self-assessment. The bourgeois principle re-examined. J. Theor. Biol. 177: 341-356.

Fisher, R.A. 1930. The genetical theory of natural selection. Oxford University. Press, Oxford.

Fretwell, S.D. and Lucas, H.L. 1969. On territorial and other factors influencing habitat distribution in birds. I. Theoretical Development. Acta Bioth., 19: 16-36.

Getz, L.L., Oli, M.K., Hofmann, J.E., McGuire, B. 2006. Vole population fluctuations: factors that initiate and determine intervals between them in *Microtus pennsylvanicus*. J. Mammal. 87: 841-847.

Hanski, I., Hansson, L., and Henttonen, H. 1991. Specialist predators, generalist predators, and the microtine rodent cycle. J. Anim. Ecol. 60: 353-367.

Hakoyama, H. and Iguchi, K. 2001. Transition from a random to an ideal free to an ideal despotic distribution: the effect of individual difference in growth. J. Ethol. 19: 129-137.

Höjesjö, J., Johnsson, J., and Bohlin, T. 2004. Habitat complexity reduces growth of aggressive and dominant brown trout (*Salmo trutta*) relative to subordinates. Behav. Ecol. Sociobiol. 56: 286-289.

Hutchinson, G.E. and MacArthur, R. 1959. Appendix: On the theoretical significance of aggressive neglect in interspecific competition. Am. Nat. 93: 133-134.

Jonzén, N., Lundburg, P., Ranta, E. and Kaitala, V. 2001. The irreducible uncertainty of the demography-environment interaction in ecology. Proc. R. Soc. Lond. B. 269: 221-225.

Jonzén, N., Wilcox, C., and Possingham, H.P. 2004. Habitat selection in temporally fluctuating environments. Am. Nat. 164: E103-E114.

Levins, R. 1962. Theory of fitness in a heterogeneous environment. 1. The fitness set and the adaptive function. Am. Nat. 96: 361-373.

Lin, Y.K. and Batzli, G.O. 2001. The influence of habitat quality on dispersal, demography, and population dynamics of voles. Ecol. Mono. 71: 245-275.

Lin, Y.K. and Batzli, G.O. 2004. Movement of voles across habitat boundaries: effects of food and cover. J. Mammal. 85: 216-224.

Maynard-Smith, J. and Parker, J. 1976. The logic of asymmetric contests. Anim. Behav. 25: 1-9.

McPeek, M.A. and Holt, R.D. 1992. The evolution of dispersal in spatially and temporally varying environments. Am. Nat. 140: 1010-1027.

Morris, D.W. 2003. Toward an ecological synthesis: a case for habitat selection. Oecologia 136: 1-13.

Morris, D.W. Diffendorfer, J.E., and Lundberg, P. 2004. Dispersal among habitats varying in fitness: Reciprocating migration through ideal habitat selection. Oikos 107: 559-575.

Morris, D.W., Lundberg, P., and Ripa, J. 2001. Hamilton's rule confronts ideal free habitat selection. Proc. R. Soc. Lond. B. 268: 291-294.

Myers, J.P., Conners, P.G., and Pitelka, F.A. 1979. Territory size in wintering sanderlings: the effects of prey abundance and intruder density. The Auk 96: 551-561.

Mylius, S.D. and Diekmann, O. 1995. On evolutionarily stable life histories, optimization and the need to be specific about density dependence. Oikos 74: 218-224.

Oatway, M.L. and Morris, D.W. 2007. Do animals select habitat at small or large scales? An experiment with meadow voles (*Microtus pennslyvanicus*). Can. J. Zool. 85: 479-487.

Posch, M., Pichler, A., and Sigmund, K. 1999. The efficiency of adapting aspiration levels. Proc. R. Soc. Lond. B. 266: 1427-1435.

Pulliam, H.R. and Danielson, B.J. 1991. Sources, sinks, and habitat selection: a landscape perspective on population dynamics. Am. Nat. 137: S50-S66.

Pusenius, J. and K. Schmidt. 2002. The effects of habitat manipulation on population distribution and foraging behavior in meadow voles. Oikos 98: 251-262.

Ranta, E. and Kaitala, V. 1999. Punishment of polygyny. Proc. R. Soc. Lond. B. 266: 2337-2341.

Ripley, S.D. 1959. Competition between sunbird and honeyeater species in the Moluccan Islands. Am. Nat. 93: 127-132.

Rodenhouse, N.L., Sherry, T.W. and Holmes, R.T. 1997. Site-dependent regulation of population size: a new synthesis. Ecology 78: 2025-2042.

Rosenzweig, M.L. 1981. A theory of habitat selection. Ecology 62: 327-335.

Rosenzweig, M.L. 1991. Habitat selection and population interactions: the search for mechanism. Am. Nat. 137: S5-S28.

Sergio, F., Blas, J., and Hiraldo, F. 2009. Predictors of floater status in a long-lived bird: a cross-sectional and longitudinal test of hypotheses. J. Anim. Ecol. 78: 109-118.

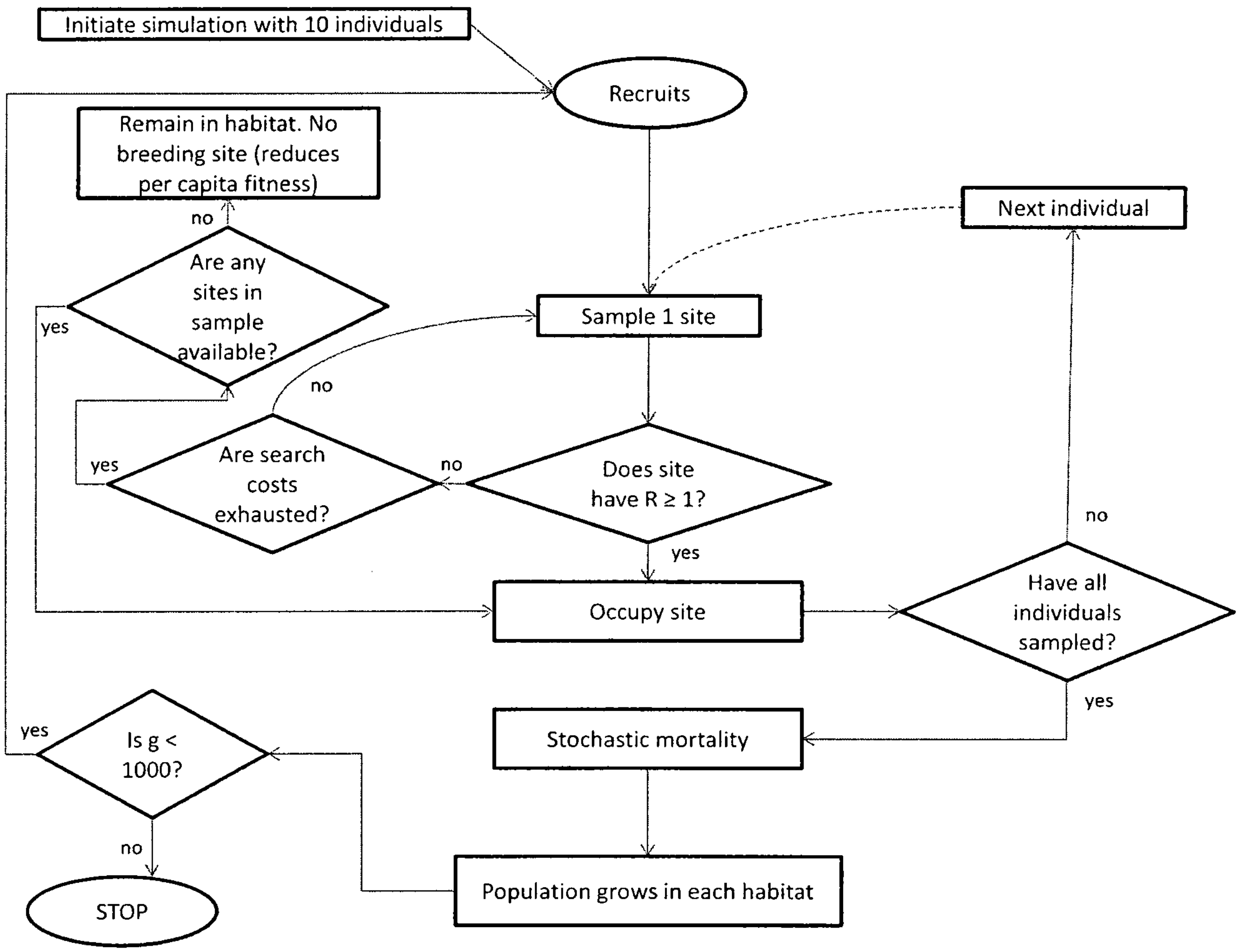
Schmidt, K.A. and Earnhardt, J.M., Brown, J.S. and Holt, R.D. 2000. Habitat selection under temporal heterogeneity: exorcizing the ghost of competition past. Ecology 81: 2622-2630.

Steneck, R.S. 2006. Possible demographic consequences of intraspecific shelter competition among American lobsters. J. Crustacean Biol. 26: 628-638.

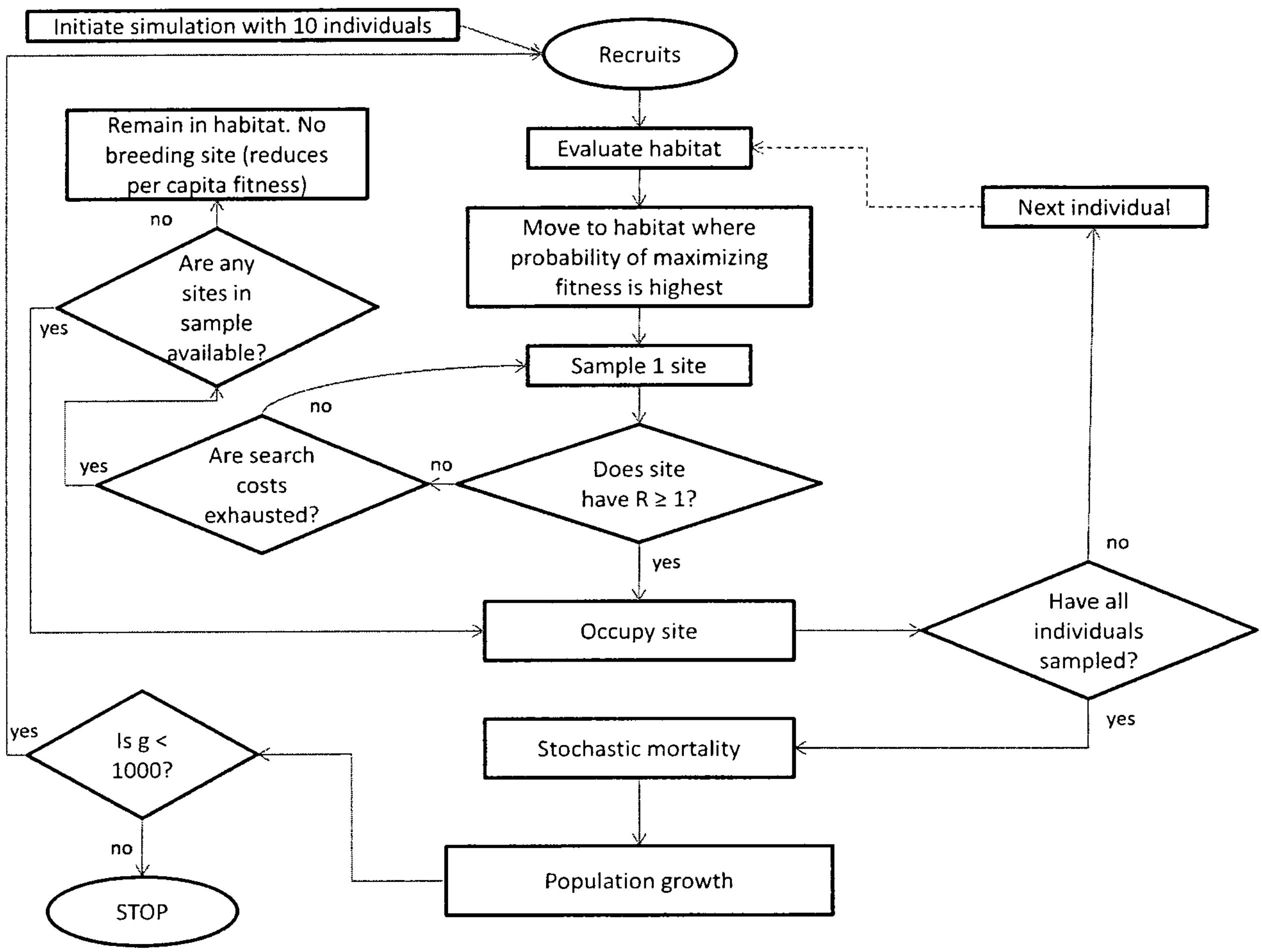
Wright, S. 1932. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In: Proceedings of the Sixth International Congress on Genetics.

Appendix 1: Flow charts of four habitat selection strategies

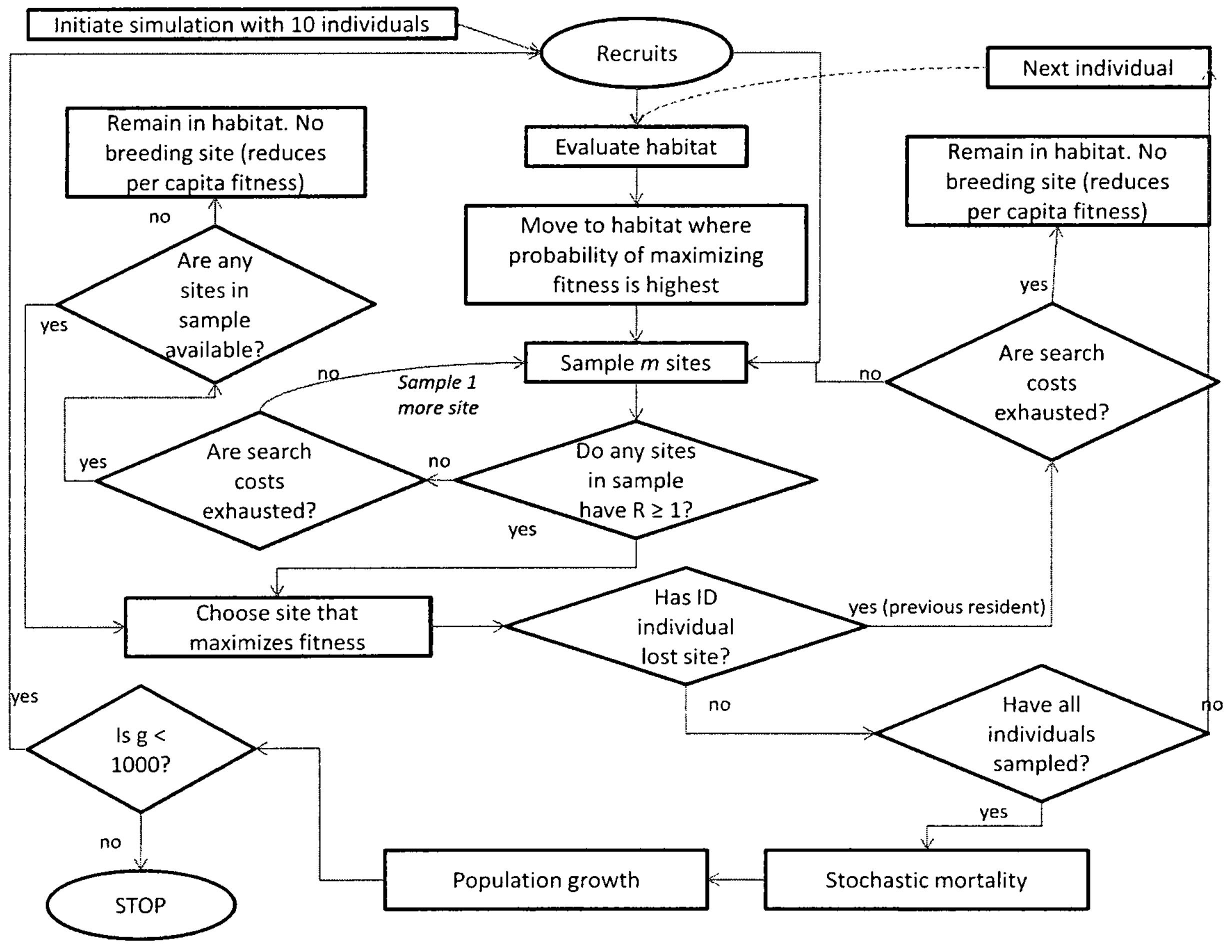
Minimal selection:



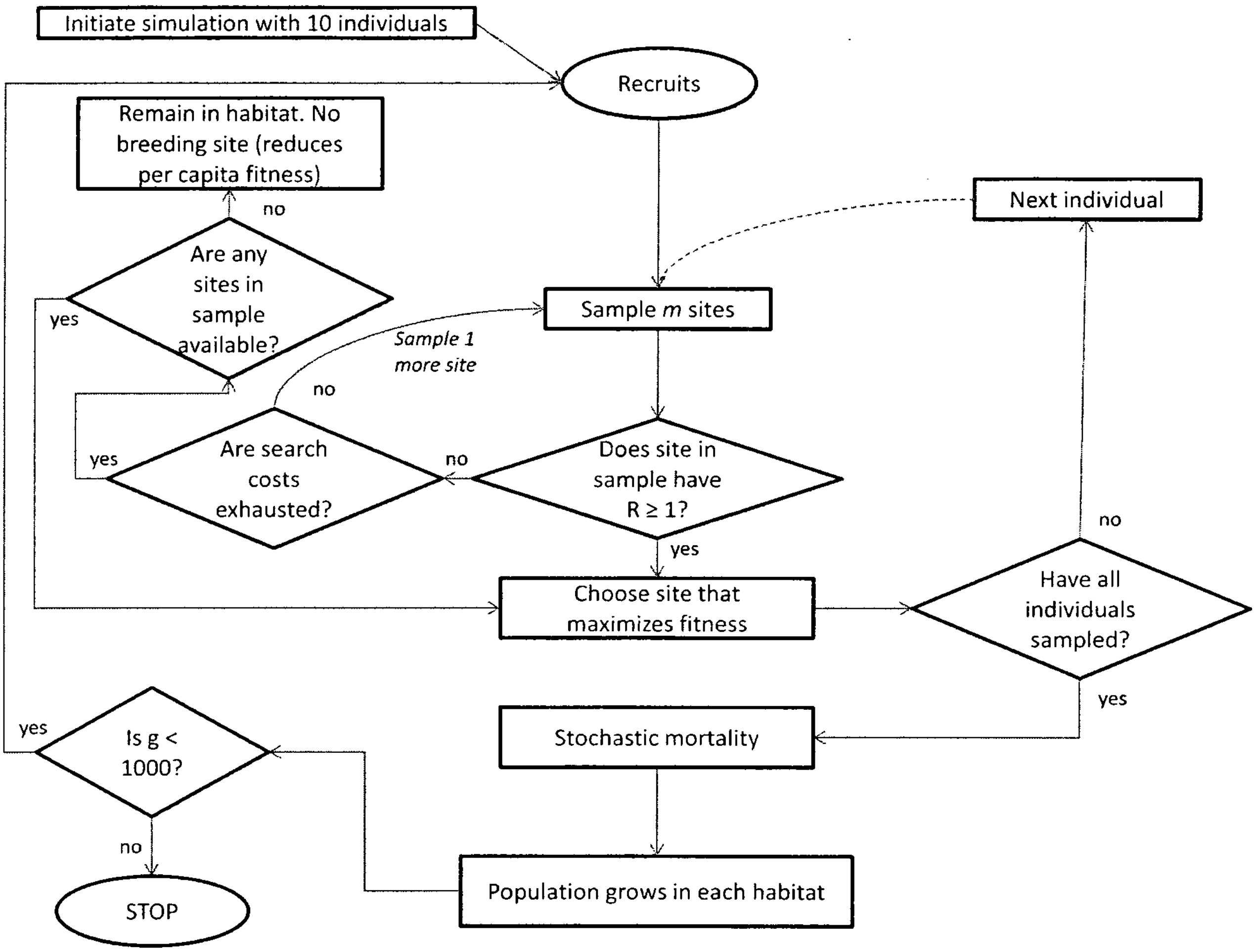
Ideal free habitat selection:



Ideal despotic habitat selection:



Ideal pre-emptive habitat selection:



Appendix 2 – Population summaries

Table A2-1: A summary of distinct simulations and replications comparing four different habitat-selection strategies.

Dataset	Distinct Simulations	Replicates per simulation	Total simulations
Differences in mean site quality	10	8	80
Differences in standard deviation	7	8	56
Sensitivity analysis	10	1	10
Total	-	-	146

Table A2-2: A summary of population dynamics from simulations comparing four different habitat-selection strategies.

Dataset	Generations	Strategy	Total number of individuals	Total number of surviving breeders in habitat A	Total number of surviving breeders in habitat B	Total number of floaters in habitat A	Total number of floaters in habitat B	Total mortality of breeders in habitat A	Total Mortality of breeders in habitat B	Total mortality of floaters in habitat A	Total mortality of floaters in habiatat B
Differences in mean site quality	80,000	**ID**	82,666,128	30,383,130	28,446,217	6,648,679	8,960,815	3,355,071	3,140,900	818,356	912,960
Differences in mean site quality	80,000	**IF**	82,460,217	29,486,175	29,873,244	7,442,192	7,437,601	3,259,432	3,304,193	826,618	830,762
Differences in mean site quality	80,000	**IP**	85,180,813	30,773,251	31,060,180	7,428,822	7,426,740	3,404,758	3,434,974	826,242	825,846
Differences in mean site quality	80,000	**MS**	80,511,118	28,825,101	28,745,991	7,452,822	7,458,355	3,189,931	3,180,314	829,890	828,714
Differences in standard deviations	56,000	**ID**	53,916,142	22,012,297	24,185,636	1,665,614	703,071	2,422,021	2,667,199	145,758	114,546
Differences in standard deviations	56,000	**IF**	55,677,710	24,004,883	24,346,989	900,175	900,443	2,644,706	2,681,316	99,617	99,581
Differences in standard deviations	56,000	**IP**	56,038,570	23,583,775	24,972,778	960,223	961,198	2,597,172	2,751,380	106,237	105,807
Differences in standard deviations	56,000	**MS**	55,675,203	23,520,100	24,726,256	952,452	952,549	2,591,581	2,720,933	105,677	105,655
Sensitivity analysis	10,000	**ID**	4,856,421	2,179,435	2,187,490	8,038	1,976	238,908	239,425	849	300
Sensitivity analysis	10,000	**IF**	2,418,861	1,086,551	1,094,419	-	-	119,048	118,843	-	-
Sensitivity analysis	10,000	**IP**	4,482,216	2,004,949	2,035,193	-	-	219,378	222,696	-	-
Sensitivity analysis	10,000	**MS**	998,310	455,092	447,170	-	-	48,702	47,346	-	-
Total	146,000		564,881,709	218,314,739	222,121,563	33,459,017	34,802,748	24,090,708	24,509,519	3,759,244	3,824,171

Appendix 3 – Limitations and recommendation for future improvements to habitat-selection models

Limitations of the simulation model

The simulation models reported here produce populations that quickly grow to, and remain near, their carrying capacities. Natural populations frequently have more variable dynamics (Ranta *et al.* 2006). It is clear from the adaptive landscape profiles (Figure 10) that fitness depends on density; and simulation results might change drastically if populations were less stable.

The ideal free model is meant to be a scramble (Nicholson 1954) whereby mean per capita fitness is depressed equally by each individual (Fretwell and Lucas 1969). In my simulations, however, I chose to model all strategies as site dependent. I modified a single site-selection algorithm to mimic all four habitat-selection strategies. In order to do this, I attributed an "aspiration level" (i.e. Posch *et al.* 1999), to individuals following minimal and ideal-free strategies. I considered several alternatives for modelling site use by IF and MS individuals. For example, allowing IF individuals to share sites, which would more closely resemble scramble competition, would not create a density-dependent response reflective of pure scramble competition because true IF individuals do not possess sites. Alternatively I might allow IF individuals to reduce mean site quality across the entire habitat. Then, assuming that carrying capacities for ID and IF individuals in identical habitats were the same, I would still need to decide on a maximum fitness and density-dependent fitness function for IF individuals. The form of the density-dependent decline in fitness would, presumably, influence output from the model. Removing site dependence from the IF strategy would introduce considerable complications for an invasion analysis. How, for example, should one compute the effects of ID individuals invading a pure IF strategy? Would IF individuals, for example, be displaced as floaters, or would they resample both habitats?

Environmental stochasticity modifies population dynamics (Ranta *et al.* 2006) and habitat selection (Jonzén *et al.* 2004). Stochastic influence in my models was slight, and should likely be increased in future simulations. I imagined that environmental stochasticity fit a uniform distribution, but others have questioned this assumption (e.g., Bell *et al.* 1993; Rohani *et al.* 2004). Increasing the stochasticity in my simulations would force populations away from carrying capacity and thus create differences in the fitness of strategies. The distribution of stochastic events might further modify population dynamics, depending on how individual strategies rebound from disturbance. Nonetheless, my simulations explore habitat selection strategies in populations near carrying capacity as might occur in stable populations. In particular, the best evidence for coexisting habitat-selection strategies comes from a population of meadow voles (Pusenius and Schimdt 2002) that was most likely above its natural carrying capacity [populations in grassland habitats in Indiana ceased population growth with lower population density Lin and Batzli 2001)].

The role of floaters

Simulated populations in my models consistently produced non-breeding floaters. Although some populations are known to support large floater populations (Blanco *et al.* 2009) below carrying capacity, floaters in my models arose mainly when populations exceeded carrying capacity. Critics might argue that floaters should have depressed fitness in proportion to density to force larger fluctuations in population dynamics. Floaters in real populations arise when breeding space is limited (Blanco *et al.* 2009), and can also dampen population fluctuations (López-Sepulcre and Kokko 2005). Nonetheless, even in rich habitats, future simulations should impose stronger density-dependent feedback on fitness.

Suggestions for improvement

Several simulation alterations would improve our insight into the evolution of habitat selection:

1. A larger density-dependent feedback of floaters would create greater fluctuations in population density through time. My simulations explored populations fluctuating near carrying capacity; larger population fluctuations would allow for a more thorough exploration of alternative scenarios.
2. Future simulations should impose a broader range of stochastic patterns in population dynamics (e.g., Bell *et al.* 1993; Rohani *et al.* 2004).
3. Future simulations should evaluate strategies with an invasion analysis (below).

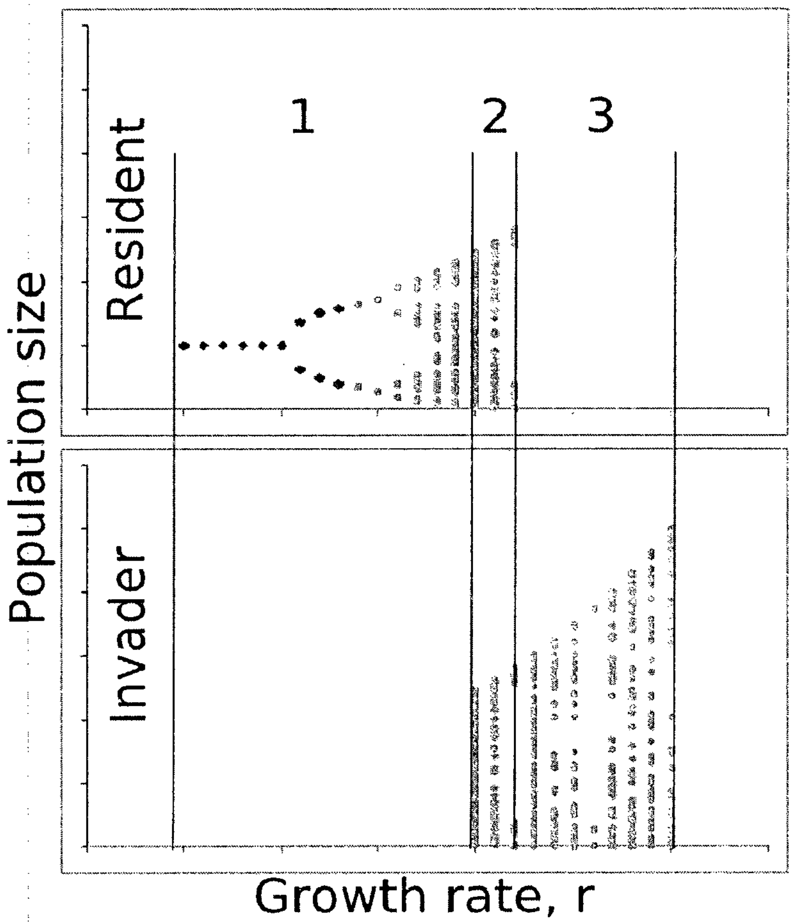
Outline of an invasion analysis

The fitness of a habitat-selection strategy depends on density and on frequency. As in other games, the evolutionary stability of a habitat-selection strategy should be thoroughly explored with an invasion analysis testing the ability of a rare mutant strategy to invade a pure strategy at its ecological equilibrium. A complete analysis of invasability would require assessing all combinations of behavioural phenotypes as both residents and invaders. The following steps outline an invasion analysis modified from Ranta and Kaitala (1999) that could be incorporated into future versions of my models.

1. Allow pure populations to grow for several generations (1000), until population dynamics stabilize.
2. Introduce a mutant with a habitat-selection strategy not yet in the established population.
3. Allow numerous generations to pass (1000).

4. Sample the population for invaders and residents for 100 generations.
5. Explore coexistence by graphing resident and invader bifurcation diagrams. Bifurcation diagrams reveal three possible scenarios (exclusion of the invader, coexistence, and invader excludes previous resident), as well as population densities (Figure. A3-1).

Figure A3-1: A example of a bifurcation diagram revealing stability of strategies across a range of growth rates, r. A population is allowed to establish over 1000 or more generations, then an invader is introduced at low density. After a thousand more generations, the population is sampled over 100 generations. The points on the graph represent attractors: stable equilibrium at low density, two-point limit cycles at moderate r values, and chaos at high r values. In zone 1, the sampling has revealed only the original population: invasion was not successful. In zone 2 and 3 there is successful invasion. Zone 2 shows coexistence (not necessarily stable) and zone 3 reveals invasion and exclusion of the resident. Modified from Ranta and Kaitala (1999).


Population size
Resident
Invader
1
2
3
Growth rate, r

Literature Cited

Bell, G, Lechowicz, M.J., Appenzeller, A., Chandler, M., DeBlois, E., Jackson, L., Mackenzie, B., Preziosi, R., Schallenburg, M. and Tinker, N. 1993. The spatial structure of the physical environment. Oecologia. 96: 114-121.

Blanco, G., Pais, J.L., Fargallo, J.A., Potti, J., Lemus, J.A. and García, J.A.D. 2009. High proportion of non breeding individuals in an isolated red-billed chough population on an oceanic island (La Palma, Canary Islands). Ardeola: 56: 229-329.

Jonzén, N., Wilcox, C., and Possingham, H.P. 2004. Habitat selection in temporally fluctuating environments. Am. Nat. 164: E103-E114.

Nicholson, A.J. 1954. An outline of the dynamics of animal populations. Aust. J. Zool. 2: 9-65.

Posch, M., Pichler, A., and Sigmund, K. 1999. The efficiency of adapting aspiration levels. Proc. R. Soc. Lond. B. 266: 1427-1435.

López-Sepulcre, A. and Kokko, H. 2005. Territorial defense, territory size, and population regulation. Am. Nat. 166: 317-329

Ranta, E. and Kaitala, V. 1999. Punishment of polygyny. Proc. R. Soc. Lond. B. 266: 2337-2341.

Ranta, E., Lundburg, P., and Kaitala, V. 2006. Ecology of populations. Cambridge University Press, Cambridge, UK.

Rohani, P. Miramontes, O. Keeling, M.J. 2004. The colour of noise in short ecological time series data. Math. Med. Biol. 21: 63-72.

Appendix 4 – Habitat isodars

Introduction and methods

Alternative strategies of habitat selection typically produce different signatures in habitat isodars (graphs of density in two adjacent habitats). Habitat isodars reveal how animals perceive and use available habitat choices (Morris 1987, 1988), and neatly illustrate the habitat selection strategy (Morris 1994). Accordingly, I regressed the density in the better-quality habitat versus that in the lower-quality habitat and fitted all isodars with both linear and quadratic models because ID and IP strategies may often produce curved isodars (Morris 1994, Knight *et al.* 2008). I removed density values of 0 from both habitats to reduce bias in the isodar slope, then used an ordinary least squares regression (SPSS v18) to "solve" the isodar (densities were measured without error). I compared the linear and quadratic models using Akaike Information Criterion scores (Akaike 1974) (R statistical software v2.7.0).

Results & Discussion

All isodars consistently produced good fit with quadratic regressions (all coefficients of variation > 0.6, Tables A4-1, A4-2, Figures A4-1, A4-2). Habitat isodars for all strategies in these simulations were curvilinear. Quadratic regressions had higher AIC scores than linear regressions in all but three cases. True ideal free populations produce linear isodars (Morris 1987, 1988). Despotism and pre-emption can produce curved isodars (Morris 1994, Knight *et al.* 2008). The site-dependent growth and aspiration level of strategies in my model led to curvilinear isodars for all strategies. MS and IP populations generally had more steeply curved isodars than IF and ID, because MS and IP sample from the entire landscape when selecting sites.

The square term changed for the isodars of all four habitat-selection strategies when the mean site quality in habitat A surpassed that in habitat B (Table A4-1, Figure A4-1).

Additionally, the square term changed for IP isodars as the mean site quality in habitat A increased (Table A4-1, Figure A4-1) because habitats supported larger populations, hence biasing the isodar toward high densities.

In general, increasing the difference in mean site quality between habitats increased the curvature of the isodars (Table A4-1, Figure A4-1); however, rich habitats (Table A4-1, Figure A4-1 $\bar{X}_a = 3$) supported large populations which tended to be more equally distributed in both habitats. Thus biased toward high densities, the isodars become linear again.

Increasing the standard deviation in site quality provided more high-quality sites and allowed faster growth rates for ID and IP strategies (not shown). A larger population growth rate biased the isodars toward high densities, and caused the square term of the IP isodar to change sign.

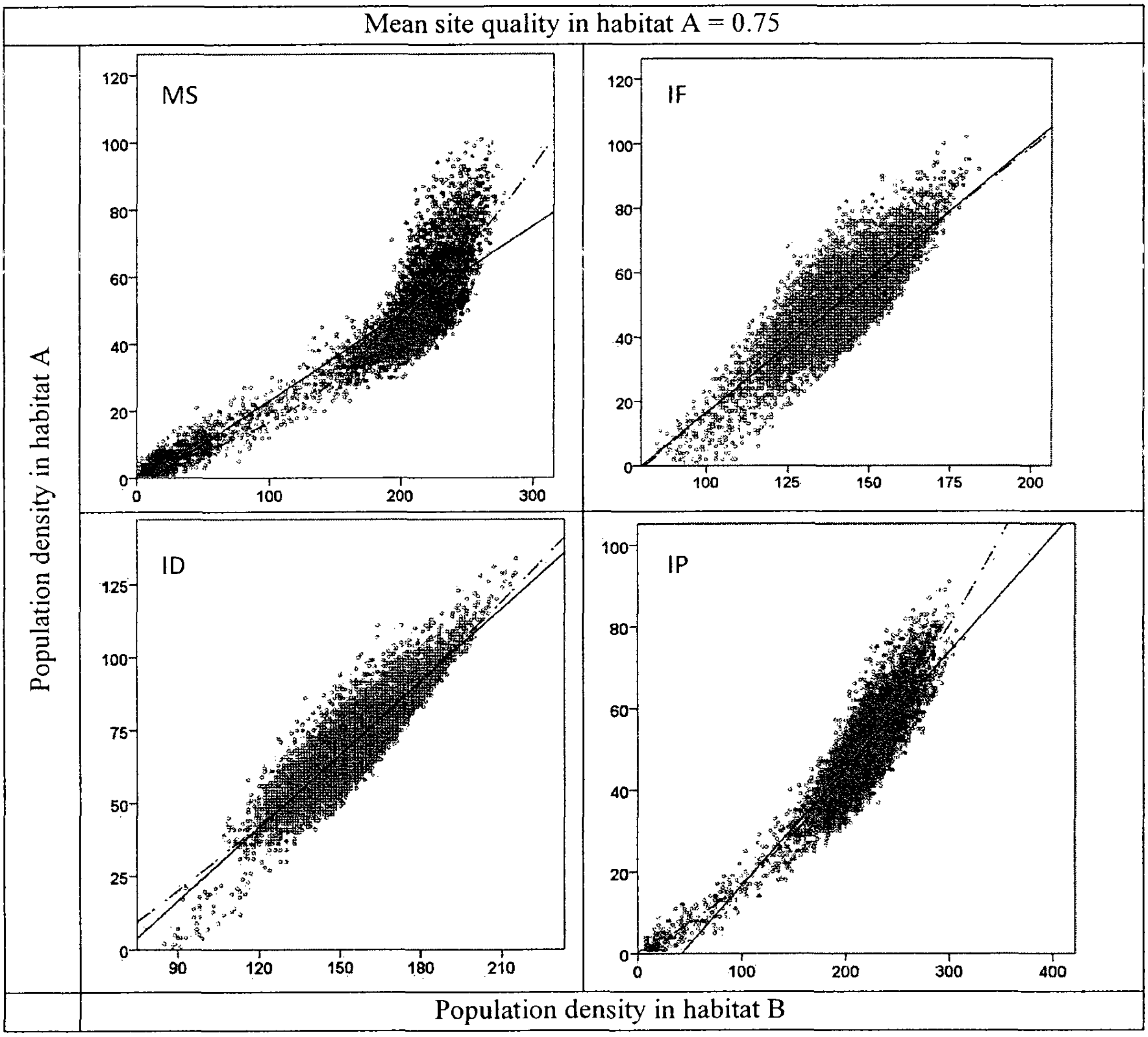
Table A4-1. Habitat isodars for simulations varying mean site quality in habitat A while site quality in habitat B remained constant. Quadratic models produced the best fit for all but one simulation. Analysis includes data from eight replications. The lower (better) of the paired AIC scores are indicated with bold type. For all simulations: $\overline{X_b} = 1$, $\sigma_a = 0.2$, $\sigma_b = 0.2$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every year) and $sv = 20$.

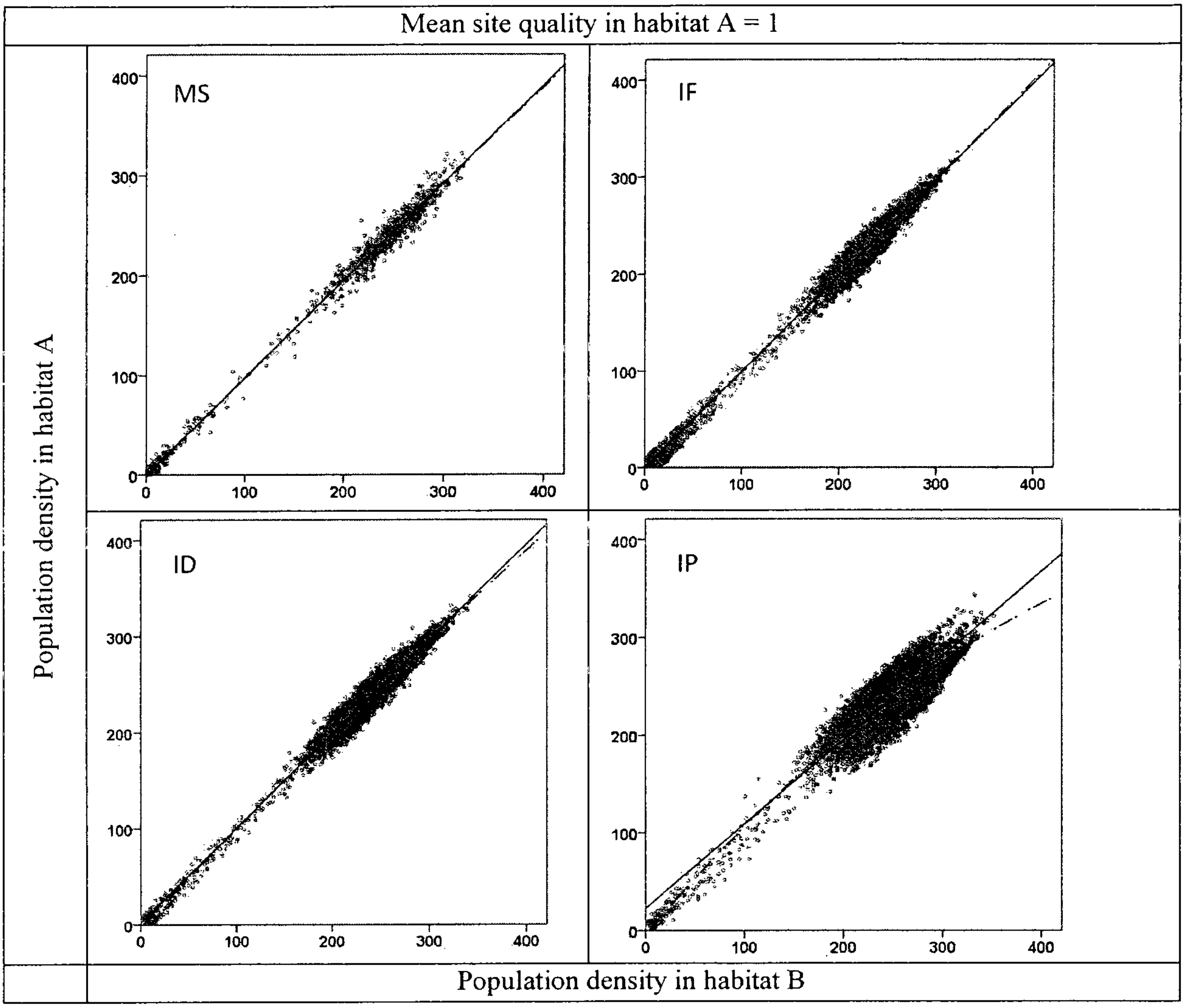
Mean site quality in habitat A	Strategy	Model	Equation	R-square	AIC
0.75	MS	Linear	$f(x) = 0.26x - 2.97$	0.864	49831.27
0.75	MS	Quadratic	$f(x) = .05x + 0.001x^2 + 3.62$	0.888	**48444.20**
0.75	IF	Linear	$f(x) = 0.83x - 66.51$	0.719	**52627.02**
0.75	IF	Quadratic	$f(x) = 0.91x - 2.9*10^{-4}x^2 - 72$	0.719	52627.86
0.75	ID	Linear	$f(x) = 0.84x - 58.70$	0.801	53524.74
0.75	ID	Quadratic	$f(x) = 0.55x + 9*10^{-4}x^2 - 36.98$	0.802	**53501.66**
0.75	IP	Linear	$f(x) = 0.29x - 11.95$	0.812	50394.72
0.75	IP	Quadratic	$f(x) = 0.45x + 8*10^{-4}x^2 + 0.47$	0.833	**49445.46**
1	MS	Linear	$f(x) = 0.98x + 0.03$	0.998	29041.73
1	MS	Quadratic	$f(x) = 0.99x - 3.59*10^{-5}x^2 + -0.01$	0.998	**29039.34**
1	IF	Linear	$f(x) = 0.99x + 0.61$	0.996	53874.59
1	IF	Quadratic	$f(x) = 0.97x + 5.11*10^{-5}x^2 + 0.761$	0.996	**53864.08**
1	ID	Linear	$f(x) = 0.98x + 2.02$	0.969	55948.56
1	ID	Quadratic	$f(x) = 1.04x - 1*10^{-4}x^2 - 2.01$	0.969	**55888.23**
1	IP	Linear	$f(x) = 0.86x + 22.87$	0.846	68179.20
1	IP	Quadratic	$f(x) = 1.18x - 0.001x^2 - 3.47$	0.856	**67643.94**
1.25	MS	Linear	$f(x) = 0.81x + 141.00$	0.744	81663.55
1.25	MS	Quadratic	$f(x) = 2.01x - 0.002x^2 + 7.72$	0.904	**73846.13**
1.25	IF	Linear	$f(x) = 0.775x + 98.85$	0.948	60262.32
1.25	IF	Quadratic	$f(x) = 0.756x + 2.96*10^{-5}x^2 + 101.69$	0.948	**60259.97**
1.25	ID	Linear	$f(x) = 0.86x + 73.02$	0.952	58762.17
1.25	ID	Quadratic	$f(x) = 0.782x + 1*15^{-4}x^2 + 85.10$	0.952	**58707.43**
1.25	IP	Linear	$f(x) = 0.58x + 233.01$	0.604	78657.07
1.25	IP	Quadratic	$f(x) = 1.71x - 0.002x^2 + 74.38$	0.776	**74104.58**
1.5	MS	Linear	$f(x) = 0.735x + 139.79$	0.792	77513.69
1.5	MS	Quadratic	$f(x) = 1.35x - 0.001x^2 + 46.80$	0.841	**75365.28**
1.5	IF	Linear	$f(x) = 0.8x + 97.35$	0.931	62436.32

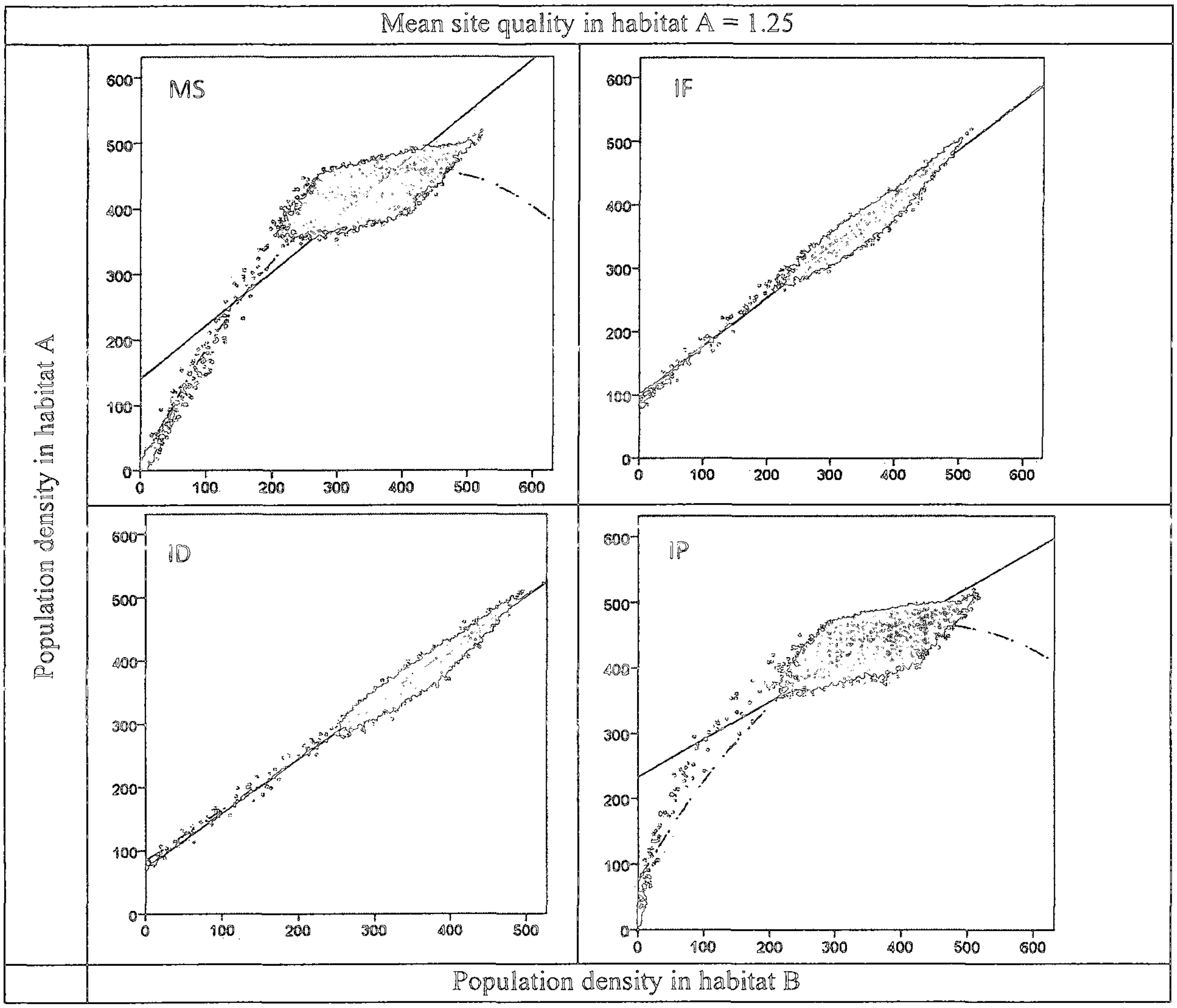
Mean site quality in habitat A	Strategy	Model	Equation	R-square	AIC
1.5	IF	Quadratic	$f(x) = 0.352x + 0.001x^2 + 186.75$	0.931	**60969.55**
1.5	ID	Linear	$f(x) = 0.85x + 82.83$	0.936	61565.09
1.5	ID	Quadratic	$f(x) = 0.445x + 0.001x^2 + 162.39$	0.943	**60651.97**
1.5	IP	Linear	$f(x) = 0.54x + 226.47$	0.671	**74794.07**
1.5	IP	Quadratic	$f(x) = 0.57x -2.86*10-5x^2 + 222.41$	0.671	74794.55
1.75	MS	Linear	$f(x) = 0.78x + 117.42$	0.854	75073.13
1.75	MS	Quadratic	$f(x) = 1.02x - 3.1*10^{-4}x^2 + 74.37$	0.86	**74727.83**
1.75	IF	Linear	$f(x) = 0.852x + 76.02$	0.939	63823.65
1.75	IF	Quadratic	$f(x) = 0.12x + 0.001x^2 + 243.98$	0.957	**61049.55**
1.75	ID	Linear	$f(x) = 0.87x + 75.43$	0.94	63851.16
1.75	ID	Quadratic	$f(x) = 0.215x + 0.001x^2 + 217.28$	0.954	**61794.80**
1.75	IP	Linear	$f(x) = 0.65x + 180.86$	0.773	73679.57
1.75	IP	Quadratic	$f(x) = -0.15x + 0.001x^2 + 350.72$	0.816	**72027.33**
2	MS	Linear	$f(x) = 0.84x + 89.32$	0.898	73534.71
2	MS	Quadratic	$f(x) = 0.85x - 2.05*10^{-5}x^2 + 85.64$	0.898	**73534.56**
2	IF	Linear	$f(x) = 0.90x + 54.56$	0.954	64131.79
2	IF	Quadratic	$f(x) = 0.052x + 0.001x^2 + 262.37$	0.969	**61045.73**
2	ID	Linear	$f(x) = 0.9x + 63.65$	0.948	64799.87
2	ID	Quadratic	$f(x) = 0.04x + 0.001x^2 + 217.28$	0.962	**62279.77**
2	IP	Linear	$f(x) = 0.769x + 125.11$	0.846	73614.04
2	IP	Quadratic	$f(x) = -0.26x + 0.001x^2 + 368.61$	0.889	**71026.61**
2.5	MS	Linear	$f(x) = 0.90x + 58.32$	0.933	73616.91
2.5	MS	Quadratic	$f(x) = 0.81x - 9.33*10^{-5}x^2 + 81.52$	0.933	**73550.09**
2.5	IF	Linear	$f(x) = 0.94x + 33.56$	0.971	64719.80
2.5	IF	Quadratic	$f(x) = 0.28x + 0.001x^2 + 215.73$	0.979	**62297.07**
2.5	ID	Linear	$f(x) = 0.94x + 44.75$	0.967	65860.12
2.5	ID	Quadratic	$f(x) = 0.27x + 0.001x^2 + 222.64$	0.975	**63626.83**
2.5	IP	Linear	$f(x) = 0.872x + 76.40$	0.907	74197.75
2.5	IP	Quadratic	$f(x) = 0.08x + 0.001x^2 + 287.15$	0.926	**72436.06**
3	MS	Linear	$f(x) = 0.94x + 36.59$	0.949	73613.59
3	MS	Quadratic	$f(x) = 0.85x - 7.7*10-5x2 + 60.93$	0.949	**73554.18**
3	IF	Linear	$f(x) = 0.97x + 21.91$	0.98	64358.03
3	IF	Quadratic	$f(x) = 0.533x + 5.8*10^{-4}x^2 + 153.01$	0.983	**63052.74**
3	ID	Linear	$f(x) = 0.99x + 21.59$	0.982	63940.75
3	ID	Quadratic	$f(x) = 0.71x + 2.3*10^{-4}x^2 + 103.53$	0.983	**63509.24**
3	IP	Linear	$f(x) = 0.925x + 48.43$	0.934	74397.58
3	IP	Quadratic	$f(x) = 0.41x + 4.2*10^{-4}x^2 + 198.14$	0.941	**73469.89**
3.5	MS	Linear	$f(x) = 0.97x + 23.32$	0.955	74335.09

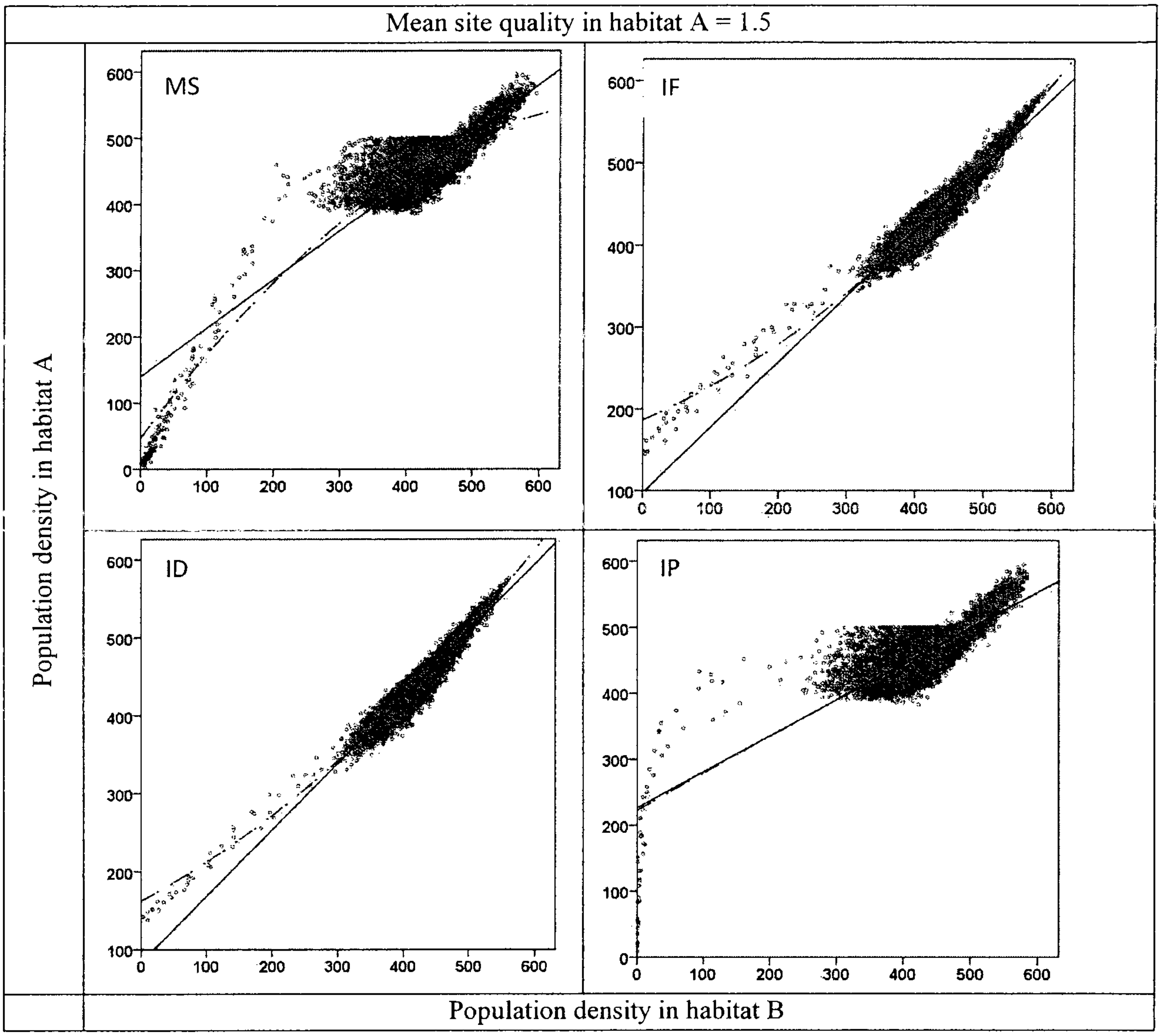
Mean site quality in habitat A	Strategy	Model	Equation	R-square	AIC
3.5	MS	Quadratic	$f(x) = 0.94x - 2.1*10^{-5}x^2 + 31.82$	0.955	**74331.23**
3.5	IF	Linear	$f(x) = 0.99x + 7.53$	0.988	62325.45
3.5	IF	Quadratic	$f(x) = 0.89x + 7.48*10^{-5}x^2 + 42.248$	0.988	**62238.07**
3.5	ID	Linear	$f(x) = 0.99x + 16.16$	0.981	65346.49
3.5	ID	Quadratic	f(x) = 0.62x + 2.8*10-4x2 + 137.81	0.984	**64327.42**
3.5	IP	Linear	$f(x) = 0.957x + 29.52$	0.942	75047.50
3.5	IP	Quadratic	$f(x) = 0.61x + 2.5*10^{-4}x^2 + 143.00$	0.944	**74704.24**
4	MS	Linear	$f(x) = 0.96x + 26.83$	0.942	76099.48
4	MS	Quadratic	$f(x) = 0.92x - 2.87*10^{-5}x^2 + 40.00$	0.942	**76091.61**
4	IF	Linear	$f(x) = 0.98x + 15.87$	0.981	65803.45
4	IF	Quadratic	$f(x) = 0.54x + 2.9*10^{-4}x^2 + 174.74$	0.984	**64461.60**
4	ID	Linear	$f(x) = 0.996x + 13.74$	0.996	65018.68
4	ID	Quadratic	$f(x) = 0.62x + 2.6*10^{-4}x^2 + 146.80$	0.619	**64260.21**
4	IP	Linear	$f(x) = 0.968x + 22.22$	0.959	75812.78
4	IP	Quadratic	$f(x) = 0.84x + 9.57*10^{-5}x^2 + 63.73$	0.959	**75742.83**

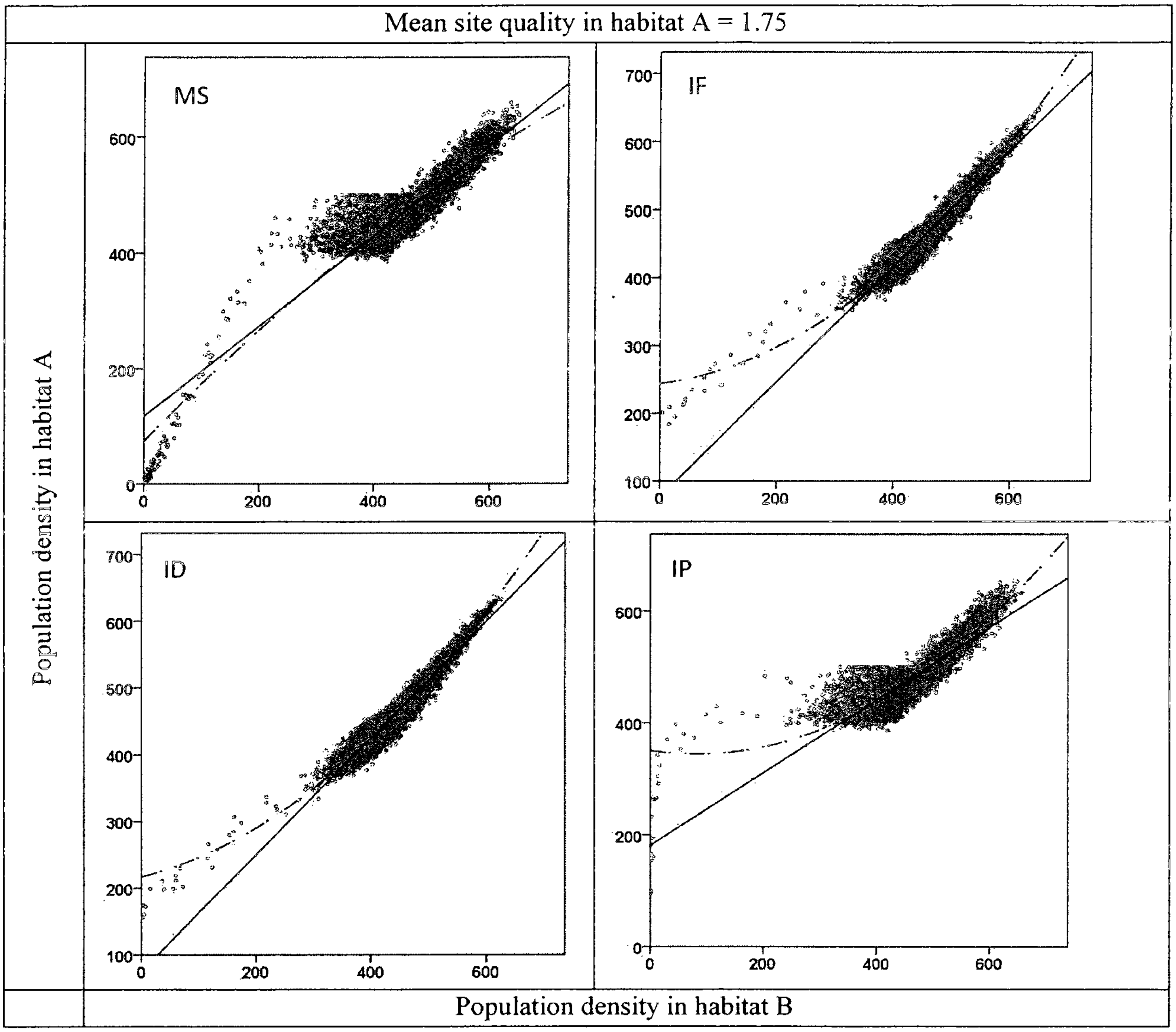
Figure A4-1. Illustrations of the isodars generated from 10 simulations of habitat selection that varied the standard deviation of site quality (Table A4-1). Each simulation was replicated 8 times.



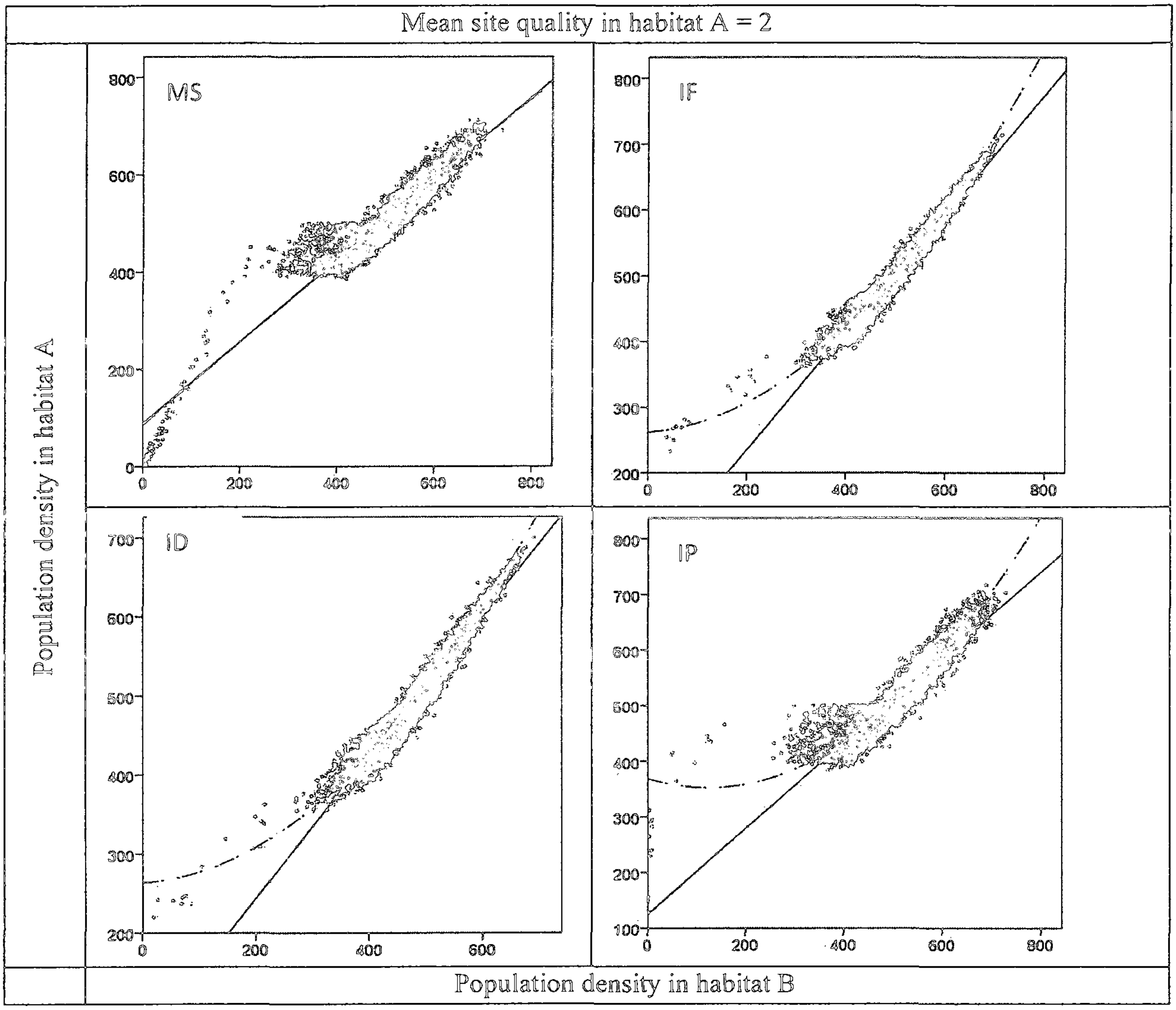

Mean site quality in habitat A = 1
MS
IF
ID
IP
0
100
200
300
400
Population density in habitat A
Population density in habitat B


Mean site quality in habitat A = 1.25
MS
IF
ID
IP
Population density in habitat A
Population density in habitat B
0
100
200
300
400
500
600

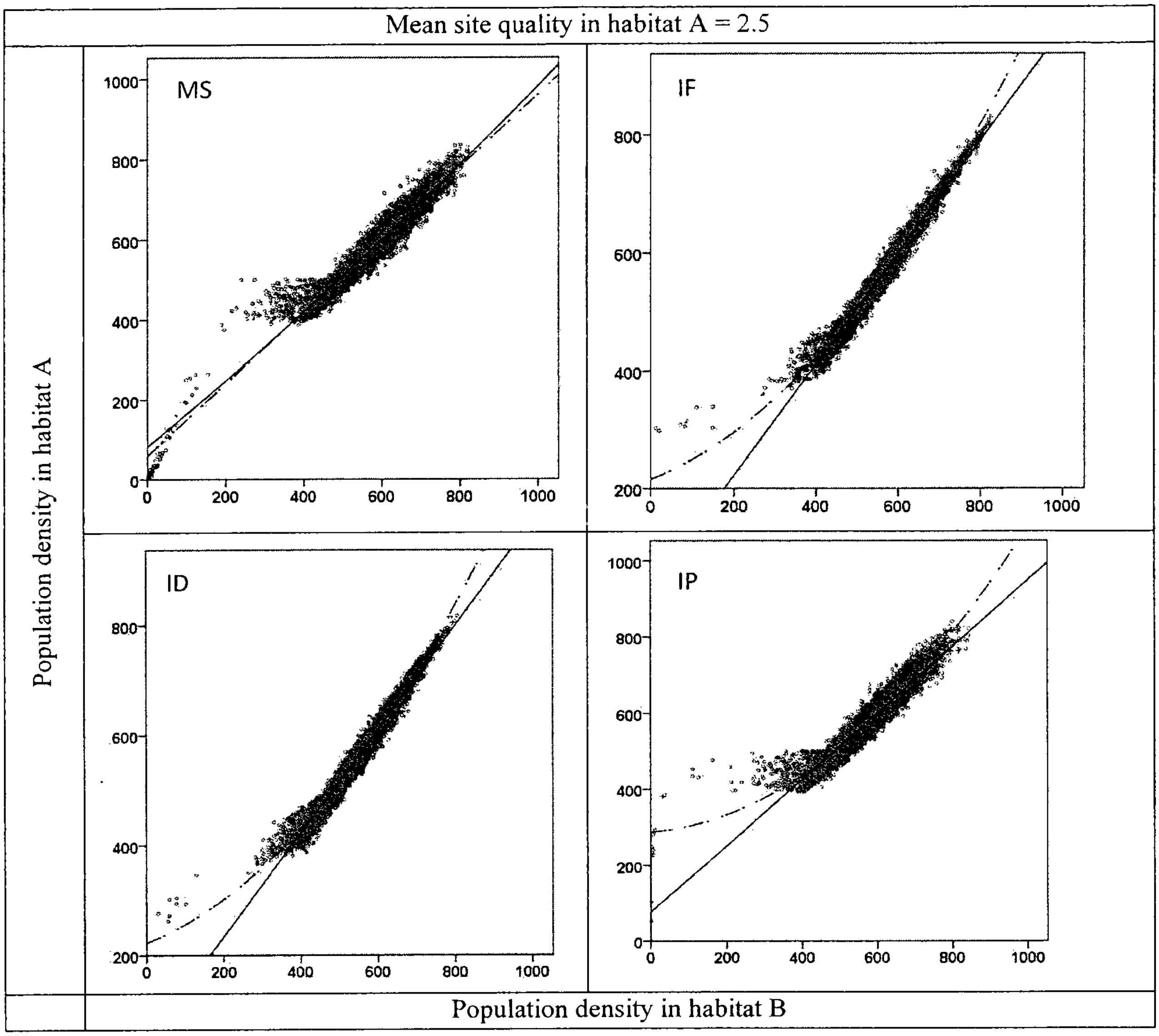

Mean site quality in habitat A = 1.5
MS
IF
ID
IP
Population density in habitat A
Population density in habitat B
0
100
200
300
400
500
600



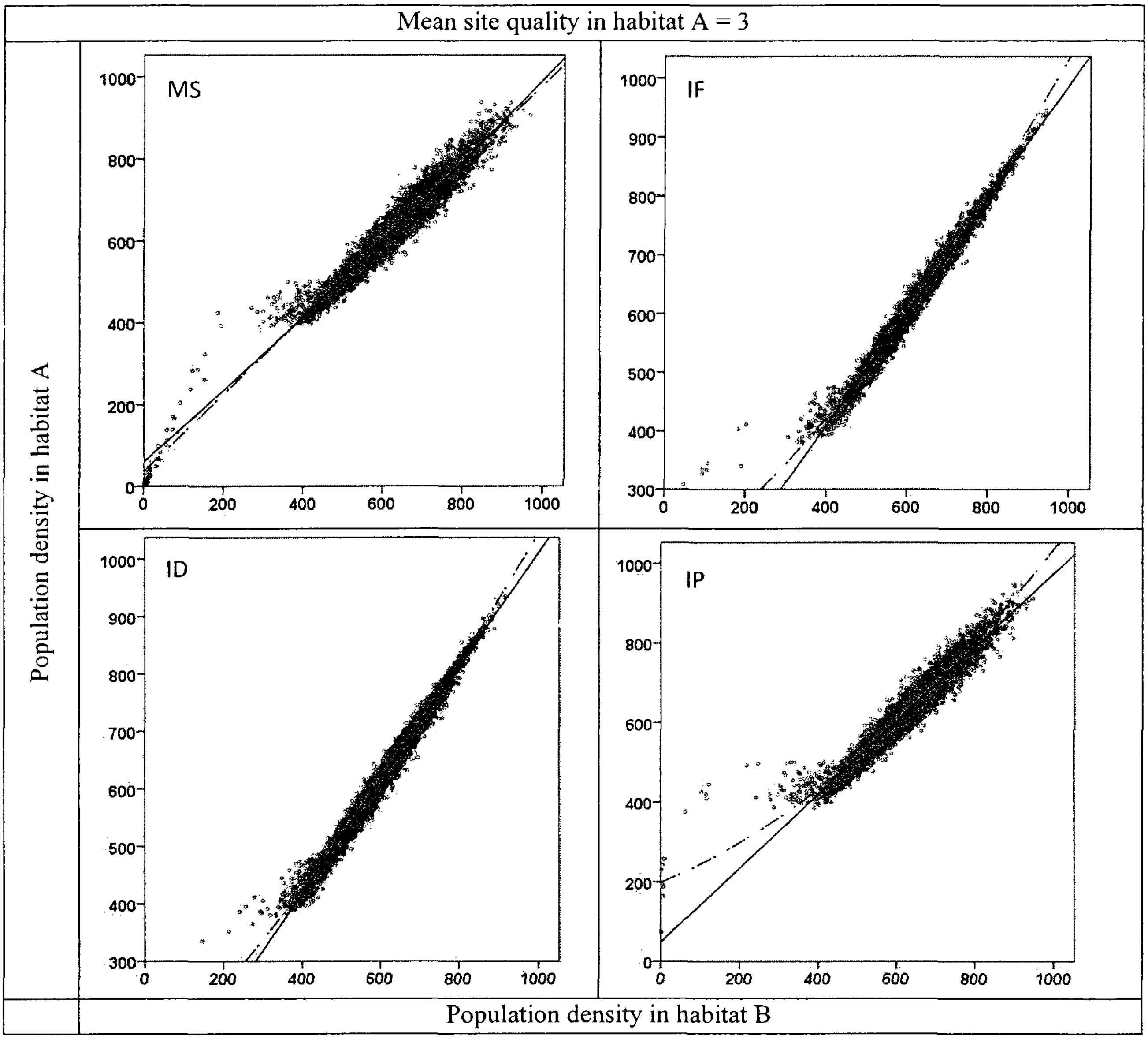
Mean site quality in habitat A = 1.75
MS
IF
ID
IP
Population density in habitat A
Population density in habitat B
0
200
400
600
700
100
300
500



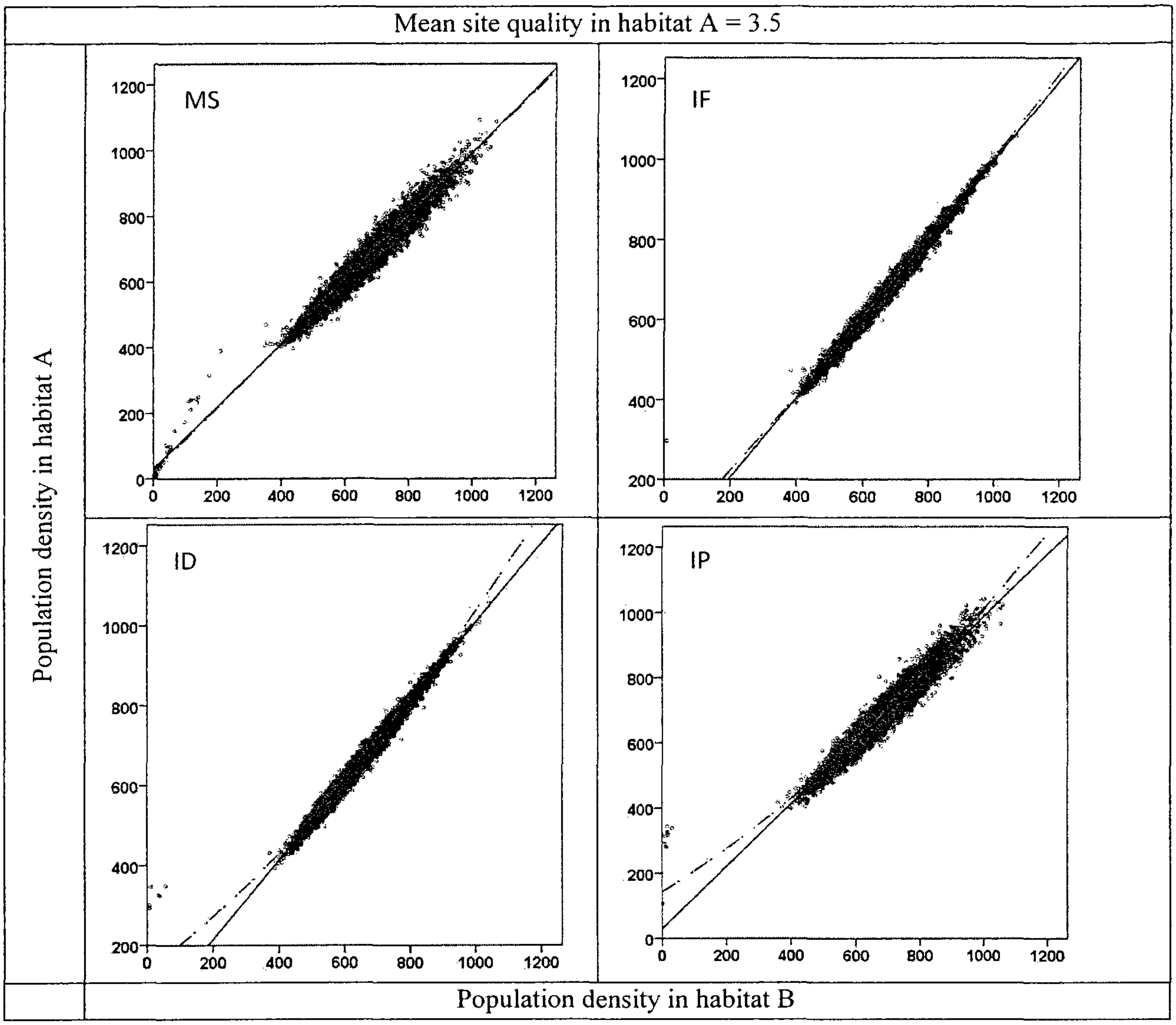
Mean site quality in habitat A = 2
MS
IF
ID
IP
Population density in habitat A
Population density in habitat B



Mean site quality in habitat A = 2.5
MS
IF
ID
IP
Population density in habitat A
Population density in habitat B
0
200
400
600
800
1000



Mean site quality in habitat A = 3
Population density in habitat A
Population density in habitat B
MS
IF
ID
IP
0
200
400
600
800
1000
300
500
700
900


Mean site quality in habitat A = 3.5
MS
IF
ID
IP
Population density in habitat A
Population density in habitat B
0
200
400
600
800
1000
1200

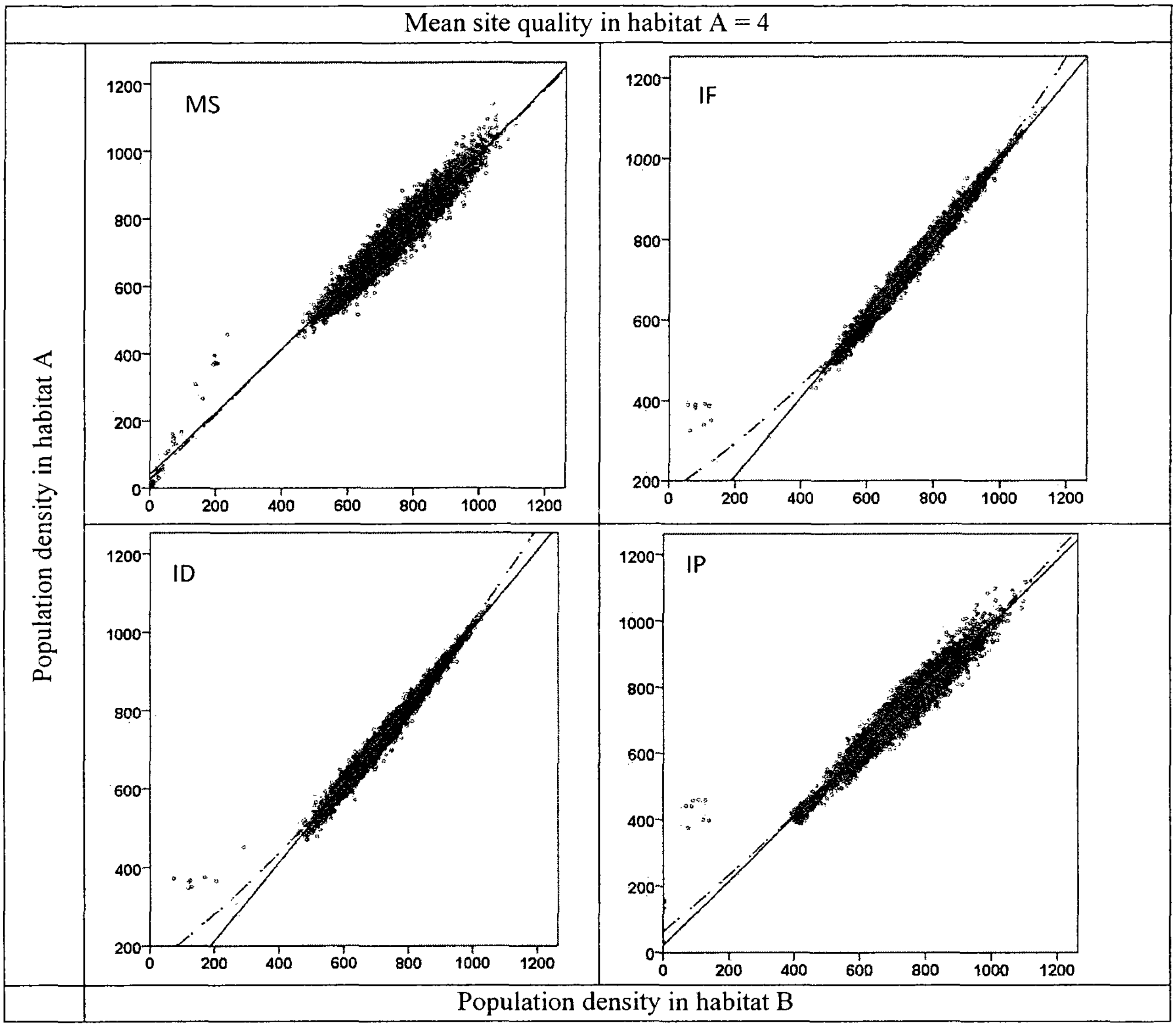
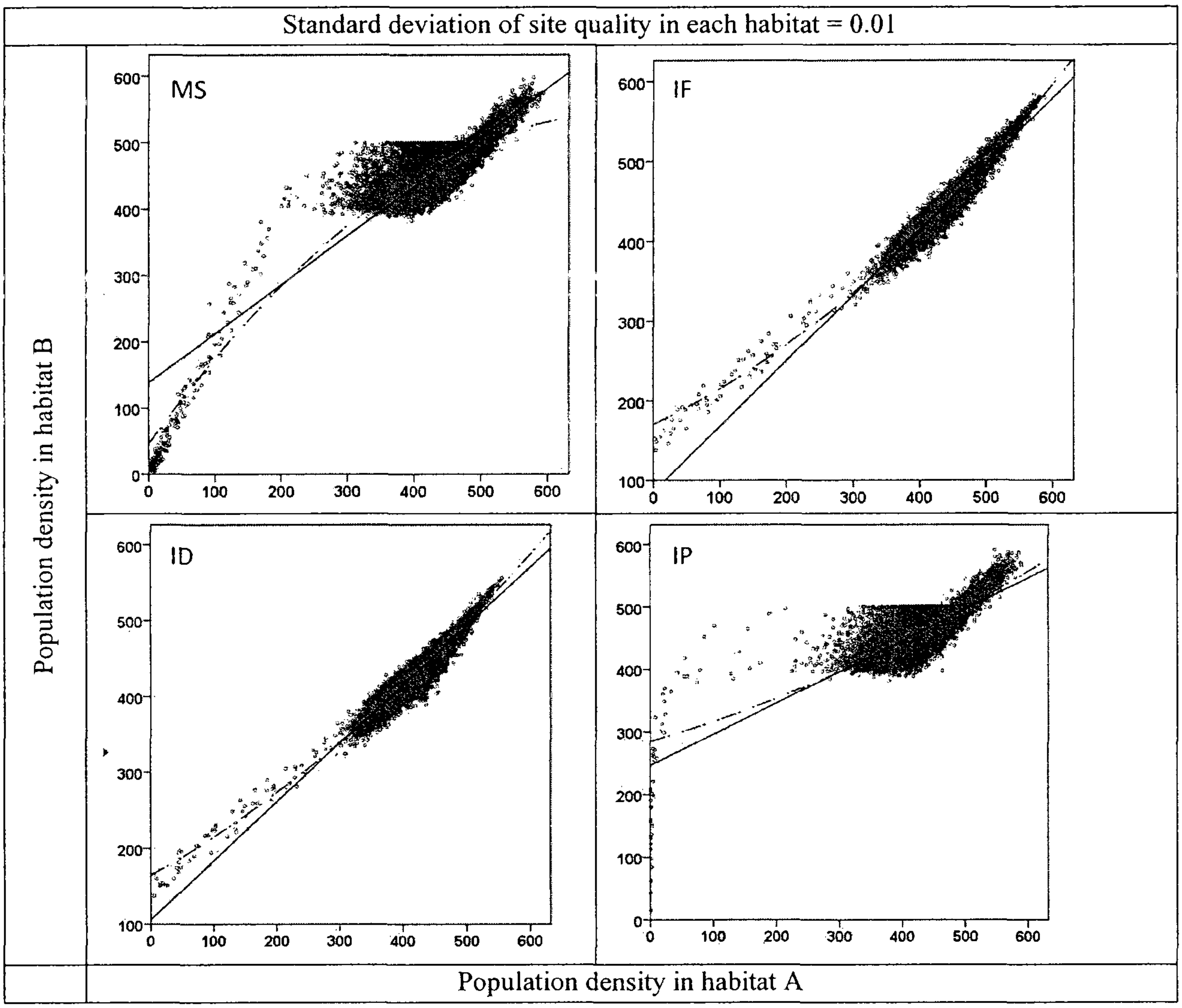

Mean site quality in habitat A = 4
Population density in habitat A
MS
IF
ID
IP
0
200
400
600
800
1000
1200
Population density in habitat B

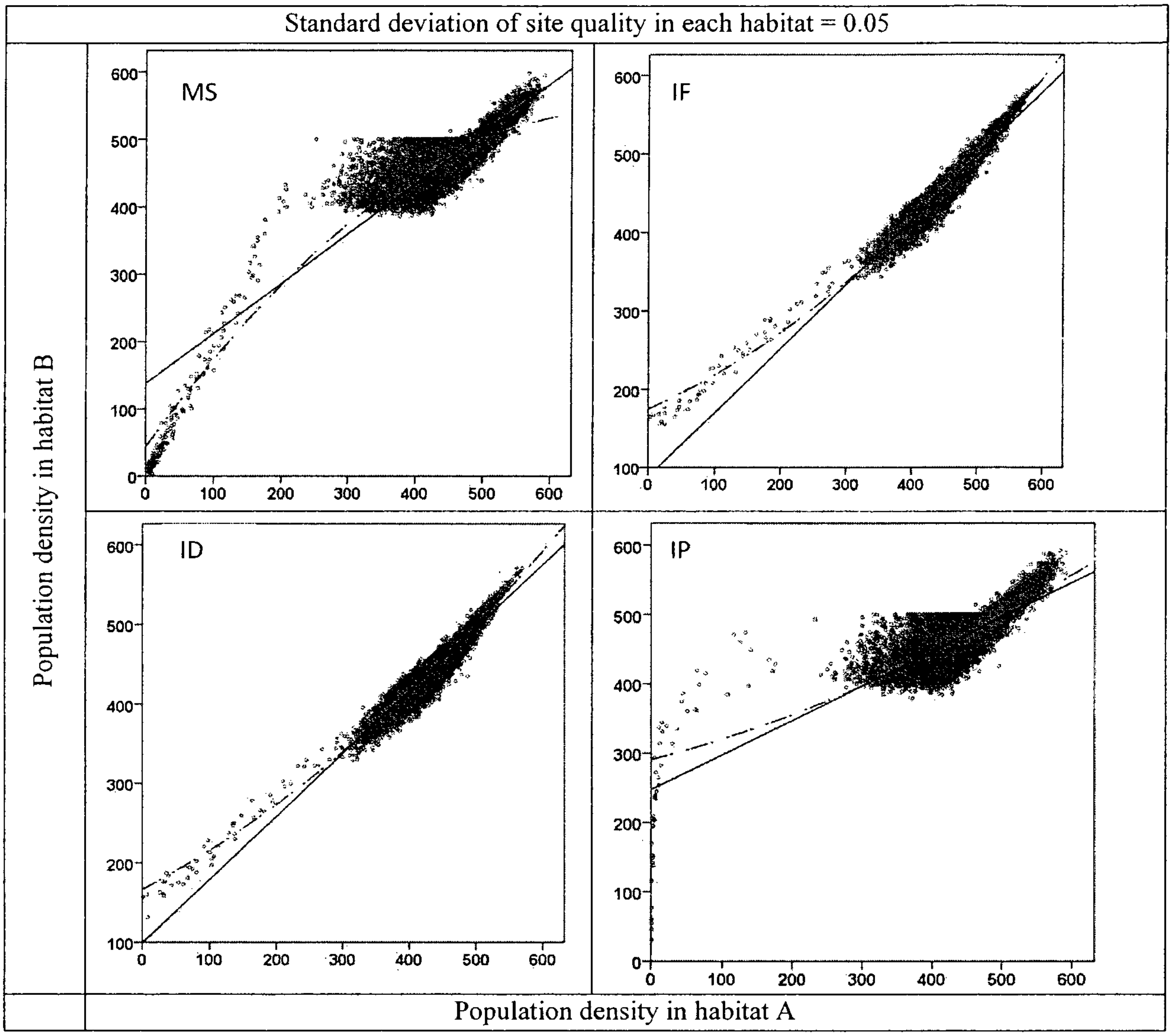
Table A4-2. Habitat isodars for simulations varying the standard deviation in site quality. Quadratic models produced the best fit for all but two simulation. Analysis includes data from eight replicates. The lower (better) of paired AIC scores are indicated with bold type. For all simulations: $\overline{X_a} = 1$, $\overline{X_b} = 1.5$, $sc = 0.001$, $dc = 0.0001$, $cc = 0.01$, $ct = 0.25$, $m = 10$, $g = 1000$, $sf = 1$ (stochastic events occur every year) and $sv = 20$.

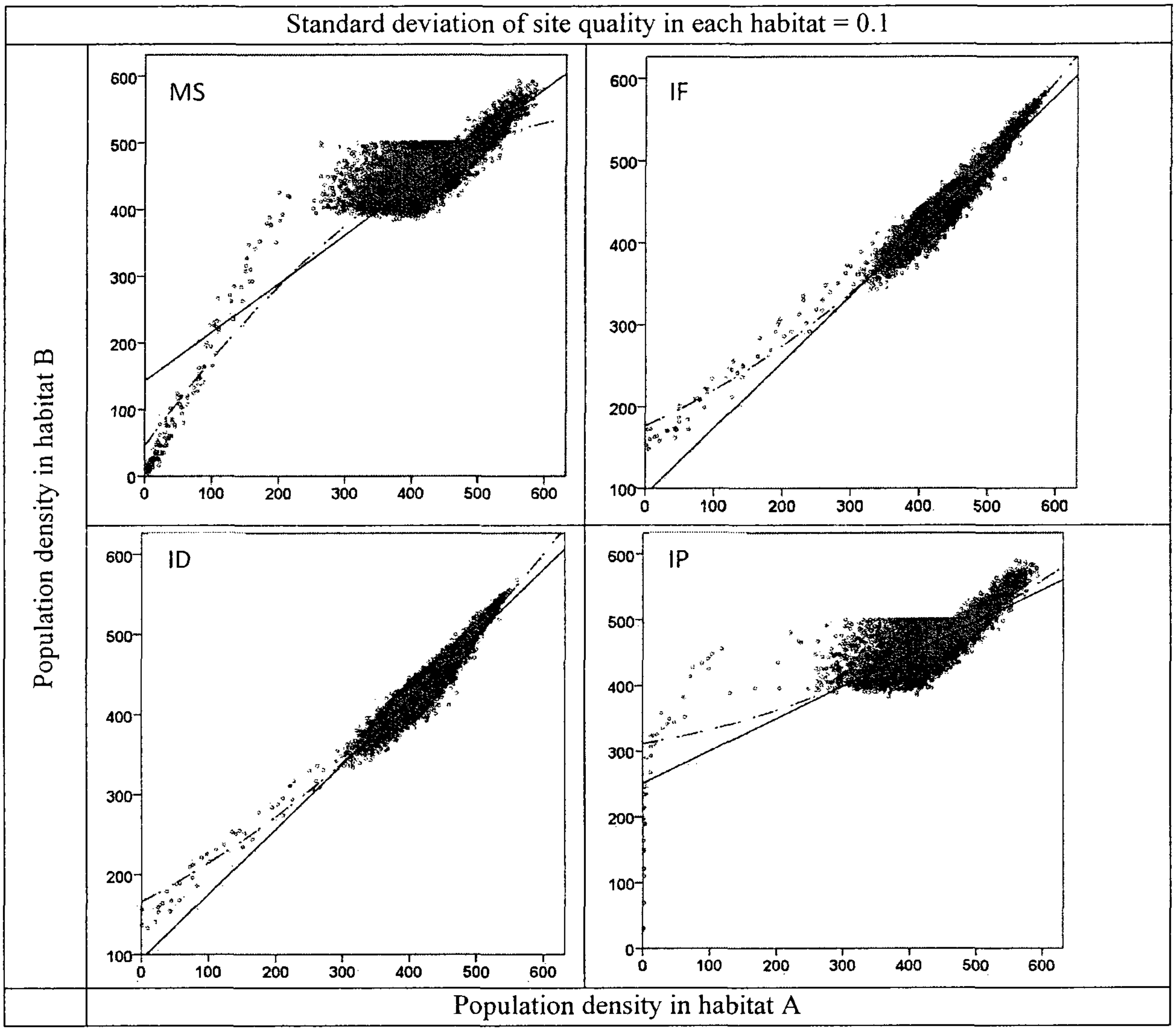
Standard deviation of site quality in each habitat	Strategy	Model	Equation	R-Square	AIC
0.01	MS	Linear	$f(x) = 0.74x + 138.39$	0.797	77934.88
0.01	MS	Quadratic	$f(x) = 1.38x - 0.001x^2 + 45.98$	0.852	**75446.12**
0.01	IF	Linear	$f(x) = 0.82x + 85.57$	0.944	61112.11
0.01	IF	Quadratic	$f(x) = 0.39x + 0.001x^2 + 170.69$	0.956	**59337.15**
0.01	ID	Linear	$f(x) = 0.77x + 106.24$	0.922	62078.59
0.01	ID	Quadratic	$f(x) = 0.46x + 4.1*10^{-4}x^2 + 164.71$	0.929	**61412.29**
0.01	IP	Linear	$f(x) = 0.5x + 246.26$	0.636	73974.30
0.01	IP	Quadratic	$f(x) = 0.5x + 2.7*10^{-4}x^2 + 285.06$	0.642	**73841.23**
0.05	MS	Linear	$f(x) = 0.74x + 138.53$	0.796	77968.92
0.05	MS	Quadratic	$f(x) = 1.39x - 0.001x^2 + 43.87$	0.852	**75440.41**
0.05	IF	Linear	$f(x) = 0.82x + 88.42$	0.942	61347.88
0.05	IF	Quadratic	$f(x) = 0.38x + 0.001x^2 + 175.22$	0.953	**59679.45**
0.05	ID	Linear	$f(x) = 0.79x + 99.39$	0.927	61836.80
0.05	ID	Quadratic	$f(x) = 0.44x + 4.5*10^{-4}x^2 + 166.76$	0.933	**61085.48**
0.05	IP	Linear	$f(x) = 0.5x + 247.52$	0.637	73948.90
0.05	IP	Quadratic	$f(x) = 0.27x + 3.0*10^{-4}x^2 + 290.62$	0.644	**73792.20**
0.1	MS	Linear	$f(x) = 0.73x + 143.50$	0.784	78236.27
0.1	MS	Quadratic	$f(x) = 1.39x - 0.001x^2 + 45.64$	0.842	**75766.54**
0.1	IF	Linear	$f(x) = 0.81x + 92.44$	0.937	61956.52
0.1	IF	Quadratic	$f(x) = 0.38x + 0.001x^2 + 176.38$	0.948	**60428.72**
0.1	ID	Linear	$f(x) = 0.81x + 94.29$	0.925	61906.59
0.1	ID	Quadratic	$f(x) = 0.43x + 4.8*10^{-4}x^2 + 166.28$	0.933	**61068.33**
0.1	IP	Linear	$f(x) = 0.5x + 251.40$	0.631	73855.80
0.1	IP	Quadratic	$f(x) = 0.17x + 4.0*10^{-4}x^2 + 311.52$	0.643	**73588.87**
0.2	MS	Linear	$f(x) = 0.74x + 139.24$	0.794	77678.31
0.2	MS	Quadratic	$f(x) = 1.37x - 0.001x^2 + 44.78$	0.846	**75327.43**
0.2	IF	Linear	$f(x) = 0.80x + 98.20$	0.932	62322.64

Standard deviation of site quality in each habitat	Strategy	Model	Equation	R-Square	AIC
0.2	IF	Quadratic	$f(x) = 0.37x + 0.001x^2 + 182.11$	0.943	**60911.23**
0.2	ID	Linear	$f(x) = 0.86x + 79.92$	0.937	61807.88
0.2	ID	Quadratic	$f(x) = 0.43x + 0.001x^2 + 162.69$	0.945	**60735.76**
0.2	IP	Linear	$f(x) = 0.5x + 228.58$	0.676	**74562.10**
0.2	IP	Quadratic	$f(x) = .56x - 1.94*10^{-5}x2 + 225.87$	0.676	74563.36
0.3	MS	Linear	$f(x) = 0.75x + 128.79$	0.828	75068.51
0.3	MS	Quadratic	$f(x) = 1.29x - 0.001x^2 + 44.70$	0.863	**73226.67**
0.3	IF	Linear	$f(x) = 0.83x + 80.71$	0.931	62407.47
0.3	IF	Quadratic	$f(x) = 0.24x + 0.001x^2 + 200.21$	0.948	**60198.33**
0.3	ID	Linear	$f(x) = 0.90x + 66.52$	0.941	61794.72
0.3	ID	Quadratic	$f(x) = 0.42x + 0.001x^2 + 161.66$	0.949	**60630.29**
0.3	IP	Linear	$f(x) = 0.61x + 196.92$	0.782	74821,62
0.3	IP	Quadratic	$f(x) = .89x - 3.8*10^{-4}x^2 + 145.55$	0.739	**74512.04**
0.4	MS	Linear	$f(x) = 0.79x + 109.63$	0.866	72869.00
0.4	MS	Quadratic	$f(x) = 1.23x - 0.001x^2 + 37.13$	0.891	**71250.84**
0.4	IF	Linear	$f(x) = 0.87x + 62.97$	0.941	61536.75
0.4	IF	Quadratic	$f(x) = 0.3x + 0.001x^2 + 178.42$	0.957	**59070.89**
0.4	ID	Linear	$f(x) = 0.93x + 63.30$	0.937	62805.22
0.4	ID	Quadratic	$f(x) = 0.48x + 0.001x^2 + 153.44$	0.942	**62078.99**
0.4	IP	Linear	$f(x) = 0.66x + 173.77$	0.775	74132.98
0.4	IP	Quadratic	$f(x) = 1.03x - 4.9*10^{-4}x^2 +106.44$	0.791	**73544.14**
0.5	MS	Linear	$f(x) = 0.82x + 91.73$	0.901	72869.00
0.5	MS	Quadratic	$f(x) = 1.17x - 4.9*10^{-5}x^2 + 34.08$	0.915	**71250.84**
0.5	IF	Linear	$f(x) = 0.88x + 55.10$	0.945	60991.33
0.5	IF	Quadratic	$f(x) = 0.33x + 0.001x^2 + 170.11$	0.957	**58979.67**
0.5	ID	Linear	$f(x) = 0.96x + 52.07$	0.947	62171.56
0.5	ID	Quadratic	$f(x) = 0.59x + 4.7*10^{-3}x^2 + 153.44$	0.952	**61537.43**
0.5	IP	Linear	$f(x) = 0.69x + 155.15$	0.814	72390.56
0.5	IP	Quadratic	$f(x) = 1.03x - 4.3*10^{-4}x^2 +93.07$	0.826	**71881.64**

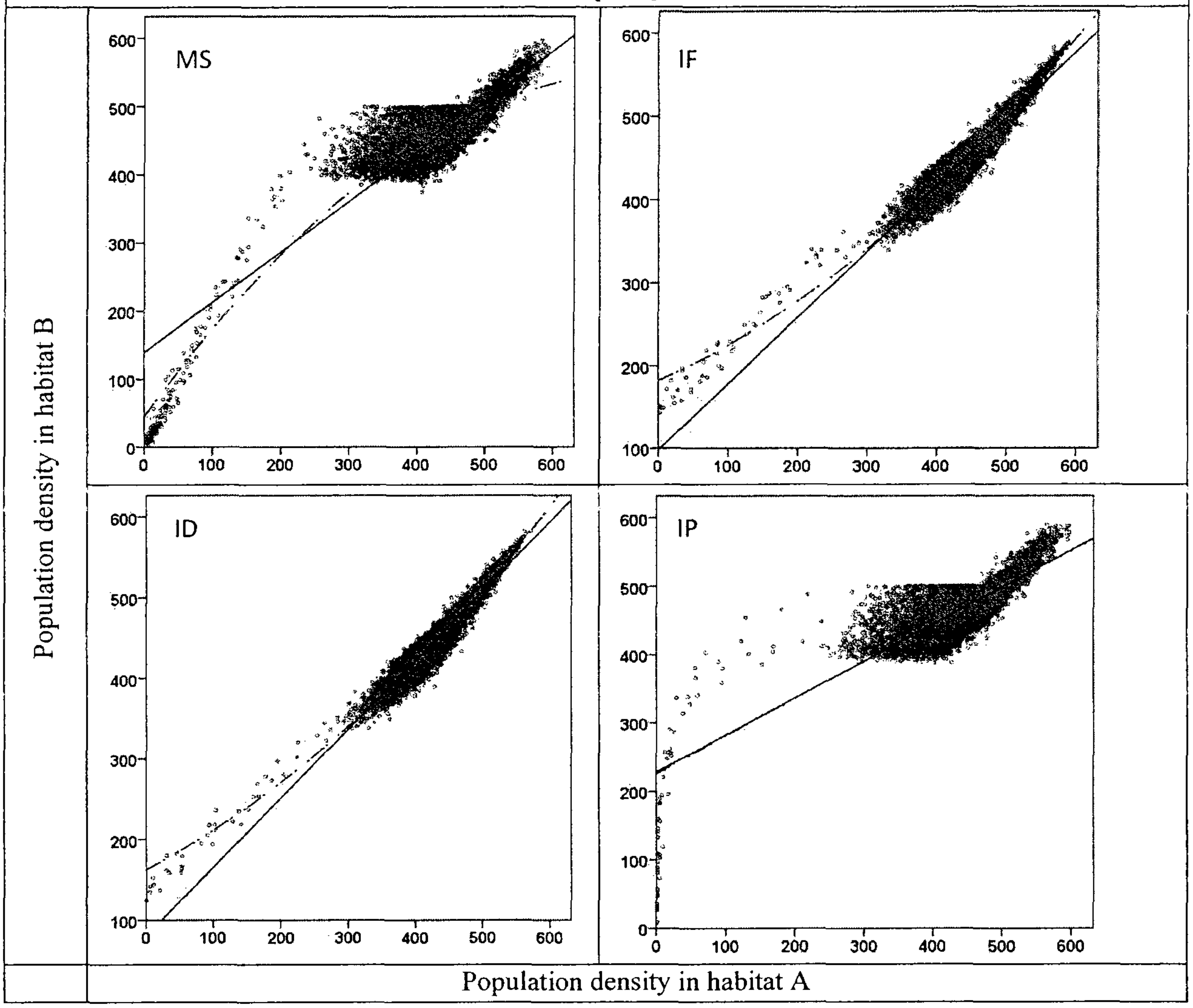
Figure A4-2. Illustrations of the isodars generated from 7 simulations of habitat selection that varied the standard deviation of site quality (Table A4-2). Each simulation was replicated 8 times.

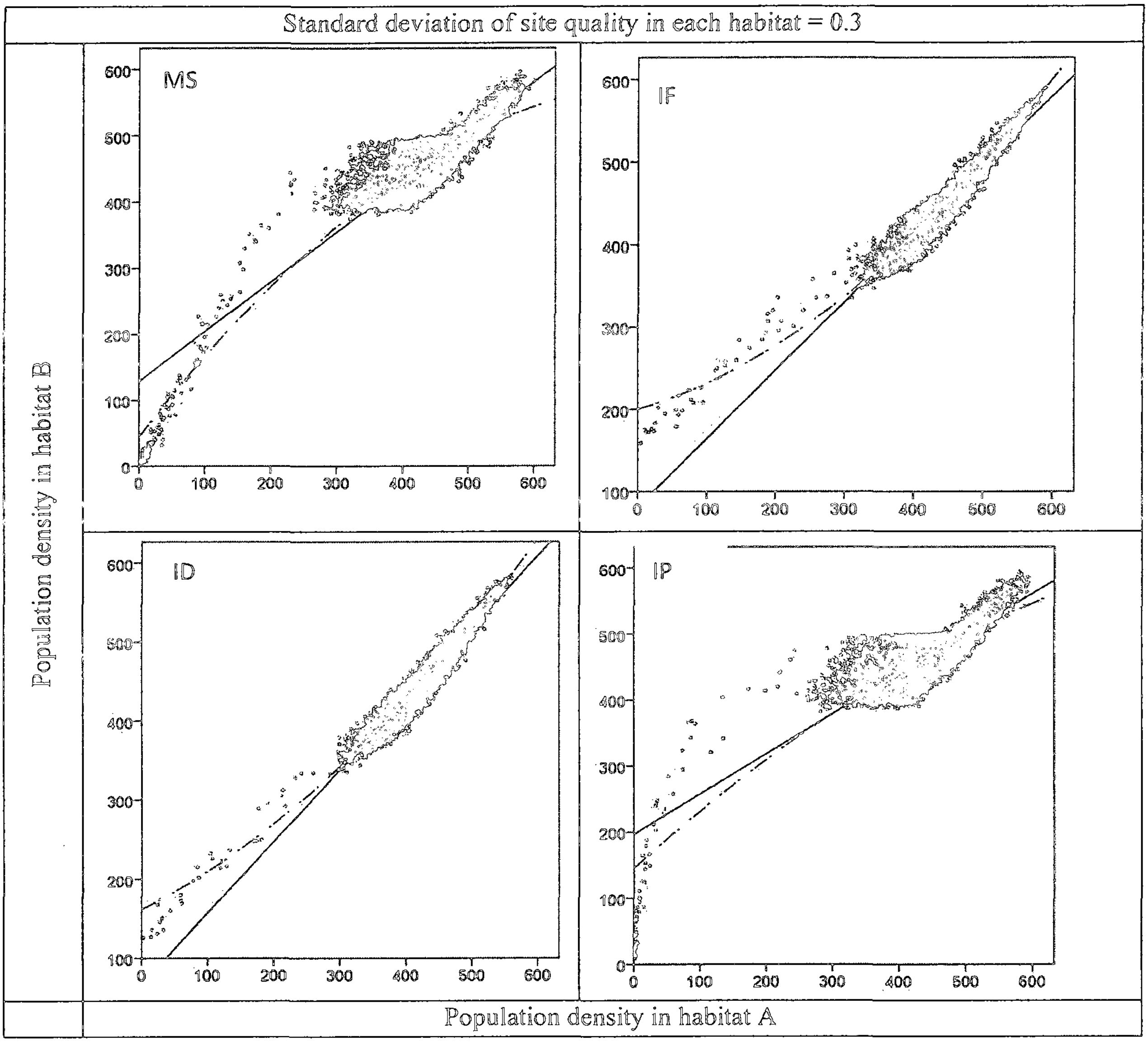


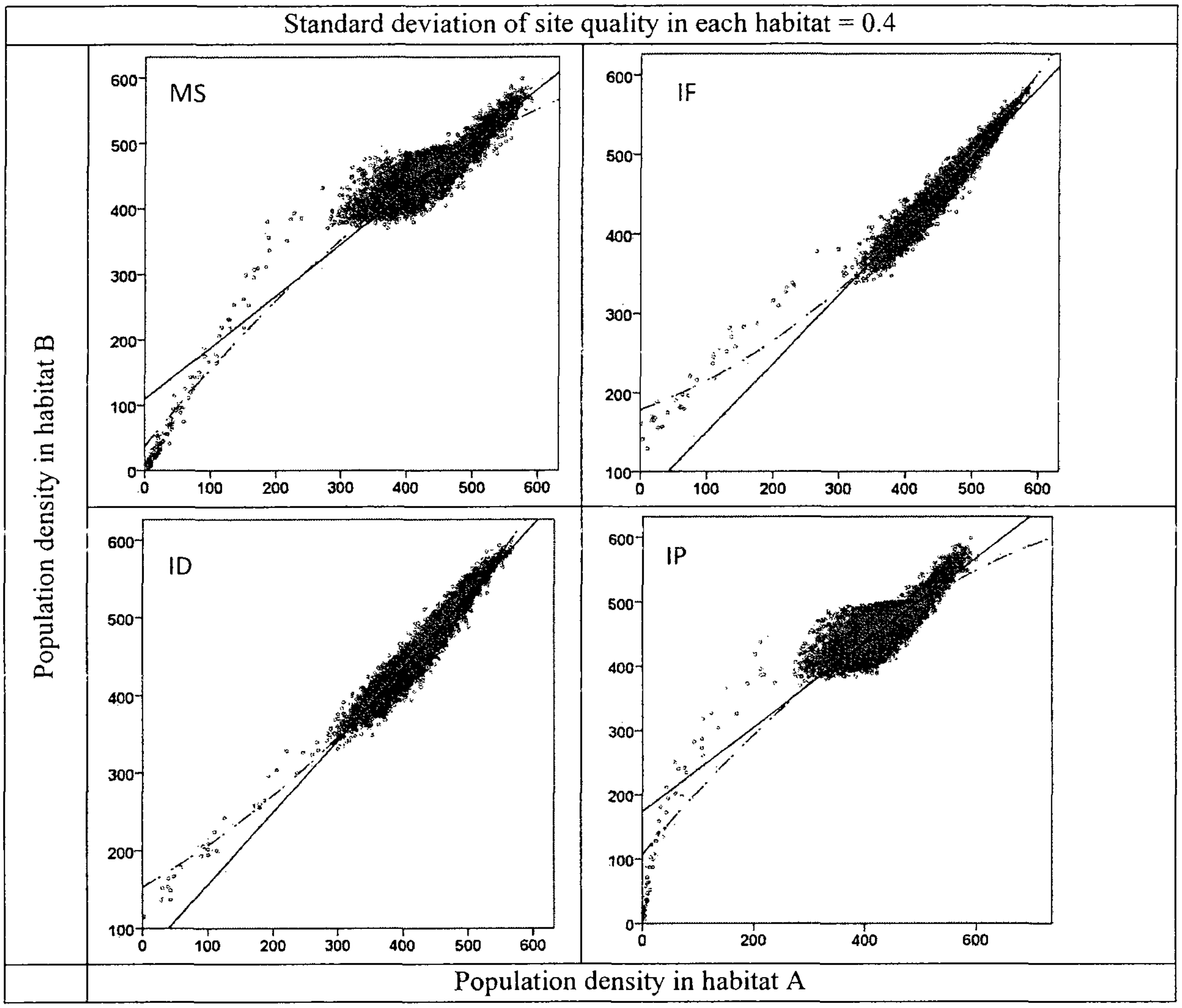

Standard deviation of site quality in each habitat = 0.05
MS
IF
ID
IP
Population density in habitat B
Population density in habitat A
0
100
200
300
400
500
600

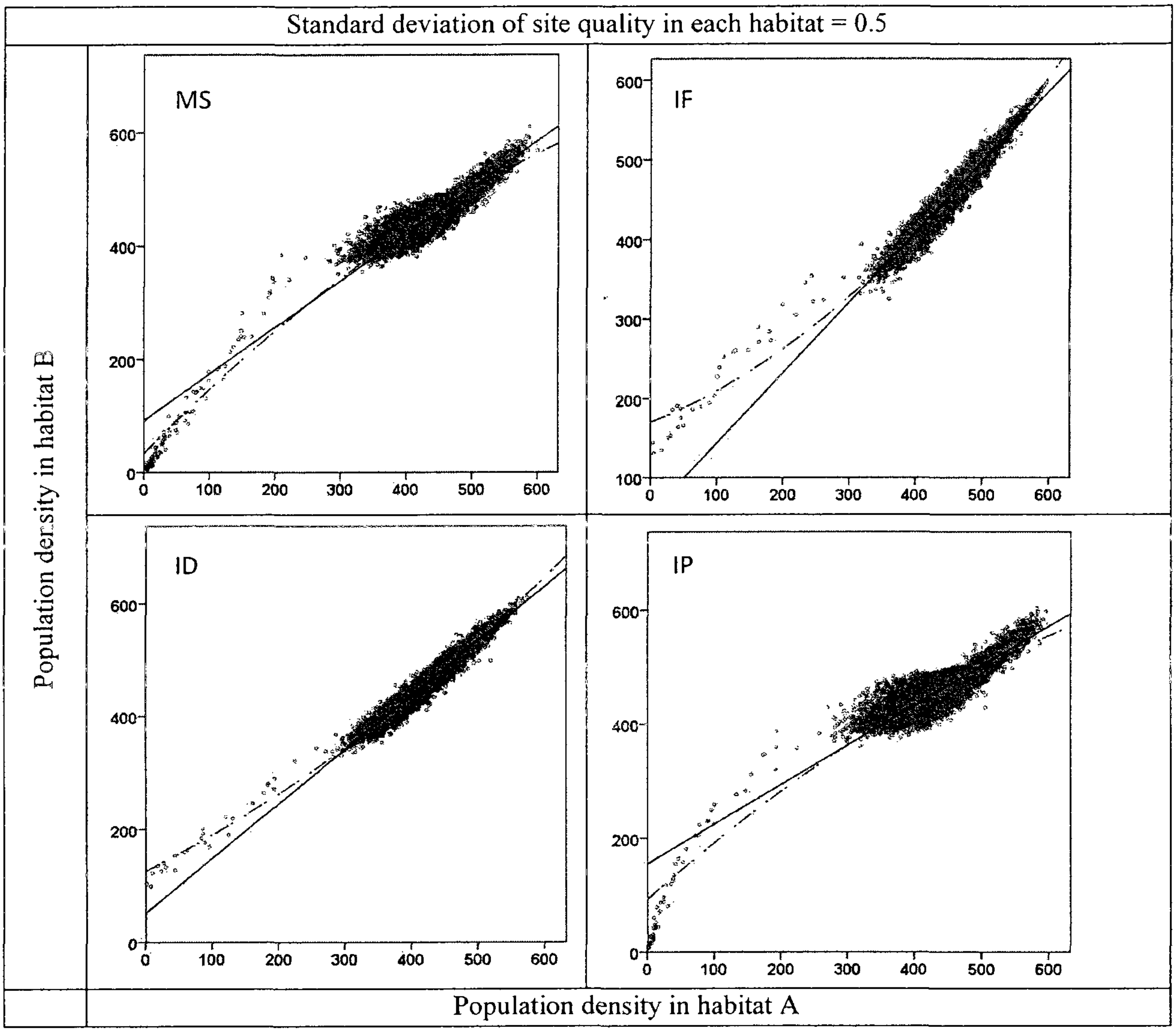

Standard deviation of site quality in each habitat = 0.1
MS
IF
ID
IP
Population density in habitat B
Population density in habitat A
0
100
200
300
400
500
600

Standard deviation of site quality in each habitat = 0.2




Standard deviation of site quality in each habitat = 0.3
MS
IF
ID
IP
Population density in habitat B
Population density in habitat A
0
100
200
300
400
500
600


Standard deviation of site quality in each habitat = 0.4
Population density in habitat B
MS
IF
ID
IP
Population density in habitat A



Standard deviation of site quality in each habitat = 0.5
MS
IF
ID
IP
Population density in habitat B
Population density in habitat A
0
100
200
300
400
500
600

References

Akaike, H. 1974. A new look at the statistical model software identification. IEEE. T. Automatic Cont.19:716-723.

Knight, T.W., Morris, D.W., and Haedrich, R.L. 2008. Inferring competitive behaviour from population census and habitat data. Isr. J. Ecol. Evol. 54: 345-359.

Morris, D.W. 1987. Tests of density-dependent habitat selection in a patchy environment. Ecol. Monogr. 57: 269-281.

Morris, D.W. 1988. Habitat-dependent population regulation and community structure. Evol. Ecol. 2: 253-269.

Morris, D.W. 1994. Habitat matching: alternatives and implications to populations and communities. Evol. Ecol. 8: 387-406.

Appendix 5 – Code for habitat-selection simulations (Windows, Python 2.5.4)

```
#habSIMStochastic.py
#Title   : Habitat Selection Evolution - Stochastic version
#Author  : Jody T. MacEachern
#Contact: jmaceach@lakeheadu.ca
#Date    : 18 Apr 2008
#Edit    : Aug 09
#Purpose: To determine which environmental conditions lead to the evolution of
#         different habitat selection strategies. Each strategy will select habitat in each
#         environment. Then strategies will be compared with a fitness maximizing
#         strategy for that environment, and their performance relative to
#         (and also occurring with) other strategies.
#         Stochastic influence affects individuals at random according to
#         two variables eFq (frequency) and eSev (severity). The latter being
#         the percentage of the population that dies.
#Written for: python 2.5.2, with python(x,y) bundle
#
#Caveat: This version has been modified from the original to fit this document. Errors may have
#been introduced in the form of spacing, indents, line wrapping without line break characters
#("\"). Etc.
#
#Key Variables:
#HabA / HabB:  Array. The site qualities of habitat A & B, respectively,
#            :  where site quality is a measure of fitness
#            :  the occupying individual will acheive (excepting costs).
#occupA / B :  List/Array. The occupany (+/-) of a breeding site in habitat A & B, respectively.
#            :   Occupancy is coded for use in the invasion analysis.
#            :  0=VACANT, 1=NULL, 2=IF, 3=ID, 4=IP.
#CostA / B  :  List/Array. The costs an individual at site i has accrued while searching for
#            :  or defending a site.
#fXa / B    :  Floaters - individuals who fail to find a breeding site yet remain in the
#            :  landscape depressing
#            :  per capita fitness. X represents the stategy: N=NULL (i.e.fNa), F=IF, D=ID, P=IP.

#SECTIONS AND DEFITIONS:
#Section 1. User entered data and model initiation.
#Section 2. Population growth, results, and stochastic event definitions
#Section 3. Habitat selection defintions
#        3.1 Habitat comparison defintion
#        3.2.1 Passive selection (null) habitat selection
#        3.2.2 Ideal free habitat selection
#        3.2.3 Ideal despotic habitat selection
#        3.2.4 Ideal pre-emptive habitat selection
#Section 4. Normal population growth. Strategies
#           grow in pure populations for x generations.
#        4.1 Passive selection (null) population growth
#        4.2 Ideal free population growth
#        4.3 Ideal despotic population growth
#        4.4 Ideal pre-emptive population growth
#Section 5. Control density analysis (i.e. adaptive landscape)
#        5.1 Fitness maximizing strategy at controlled densities
#        5.2 Passive selection performance at controlled densities
#        5.3 Ideal free performance at controlled densities
#        5.4 Ideal despotic performance at controlled densities
#        5.5 Ideal pre-emptive performance at controlled densities
#Section 6. Invasion Analysis. Tests ability of best strategy from growth phase
#           to resist invasion from other strategies.
#        6.1 IFD is best strategy
#        6.1.1 Introduce IDD
#        6.1.2 Introduce IPD
#        6.1.3 Introduce IDD & IPD
#        6.2 IDD is best strategy
#        6.2.1 Introduce IFD
```

```
#       6.2.2 Introduce IPD
#       6.2.3 Introduce IFD and IPD
#       6.3 IPD is best strategy
#       6.3.1 Introduce IDD
#       6.3.2 Introduce IFD
#       6.3.3 Introduce IDD and IFD

#DEFINTIONS
#habsel    : Initializes the model. Draws site qualities from distributions according
#          : to user-entered parameters. Initializes output folder, parameter text file
#          : and growth phase file. Stores landscape (habA/habB) as pickle files.
#popgrowth : Takes the habitat, occupancy, cost vectors, and "floaters" and returns
#          : population growth for a given habitat.
#Results   : Takes the habitats, occupancy and cost vectors and floaters and returns a summary
#          : returns a summary of costs and fitness in the landscape (for output).
#habcomp   : Compares the two habitats based on total site quality and probability of getting
#          : a site.
#          : used by ID and IF strategies.
#StochasticEvent : Takes the floaters, cost, and occupancy vectors and applies a user-defined
#          : stochastic mortality event. Parameters are frequency (1 in x generations,
#          : where x is drawn from a uniform
#          : distribution, and severity (y drawn from a uniform distribution).
#NULL      : Habitat selection by passive selectors - chooses the first empty site in the
#          : landscape
#          : where the site quality is equal to or greater than replacement.
#IF        : Ideal free habtitat selection. Chooses the first unoccupied site in the best
#          : habitat where the site quality is equal to or greather than replacement.
#ID        : Ideal despotic habitat selection. Takes a minimum sample of sites from the best
#          : habitat
#          : and chooses the best. Can also take occupied sites. This definition also handles the
#          : ousted individual (including potentially different strategies during the invasion
#          : analysis).
#IP        : Ideal pre-emptive habitat selection. Takes a minimum sample of sites from the
#          : landscape,and chooses the best unoccupied site that it finds.

#Directions for usage, reads on input:
print """Habitat selection algorithm.
-----------------------------------------------------------------
Required input:
sc     = the cost of sampling a site
dc     = cost to resident when defending a site from a sampling individual
cc     = cost of taking an occupied site (IDD-IDD)
cthres = the maximum cost an individual will pay while searching for a site
habsef = the minimum number of vacant sites a despotic or pre-emptive individual will sample
gen    = the number of generations to run for (default 1000)
eSev   = max. of a uniform distribution defining the severity (as % mortality) of stochastic
influence.
eFq    = max. of a uniform distribution defining frequency of stochastic events. Set improbably
large for
deterministic simulation. Set to 0 for annual stochastic event.

>>>>> HABITAT PARAMETERS <<<<<
hab1 and hab2 are the habitat parameters, take the form [x,x,x]
NORMAL DISTRIBUTION           [1,mean,standard deviation]
EXPONENTIAL DISTRIBUTION      [2,beta,n/a]
UNIFORM DISTRIBUTION          [3,min,max]"""

###############################################################################################
#SECTION 1
#User entered data and site quality distributions
###############################################################################################
```

```
#Required modules. External module numpy, rest are native modules.
from numpy import *             #Math, stats on arrays and matrices
from scipy import *
#For random number generation, copying arrays/matrices
import random, copy, time, os, cPickle, math
def habsel(hab1, hab2, gen=1000, sc=0.01, dc=0.001, cc=0.05, cthres=1, habsef=10, eSev=20,\
eFq=0):
    """Habitat selection definition. Stochastic version. Runs the habitat selection algorithm.
Takes
       model parameters as arguments. This way you can queue up multiple iterations
       of the whole program, to, for example, compare parameter values or performance over
       specific habitats."""

    #Specifies variables as global for use in sub definitions.
    global habA, habB, occup, occupFA, occupFB, occupNA, occupNB, occupDA, occupDB, occupPA,\
    occupPB, cost, costNA, costNB, costFA, costFB, costDA, costDB, costPA, costPB, sampler,
    samplerP

    random.seed()

    #querying sys time, getting sys path
    path = os.path.join(os.getcwd(), (str(time.localtime()[1]) + '_' + str(time.localtime()[2])\
    + '_' + str(time.localtime()[3]) + '_' + str(time.localtime()[4])))

    os.makedirs(path)
    fnum = file(os.path.join(path, "growth.csv"), "w")  #create the growth output

    i = 0
    habA = []
    habB = []
    while i < 2:
        #Now fill a list with sites according to site quality distribution.
        if i == 0:
            d = hab1[0]
            if d == 1:                      #Normal
                meanD = hab1[1]
                stdD  = hab1[2]
            elif d == 2:                    #Exponential
                beta = hab1[1]               #beta
            elif d == 3:                    #Uniform
                a    = hab1[1]               #min
                beta = hab1[2]               #max
        else:
            d = hab2[0]
            if d == 1:
                meanD = hab2[1]
                stdD  = hab2[2]
            elif d == 2:
                beta = hab2[1]
            elif d == 3:
                a    = hab2[1]
                beta = hab2[2]

        j = 0
        while j < 500:
            if d == 2 or d == 3:

                if d == 2:
                    r = random.expovariate(beta)
                elif d == 3:
                    r = random.uniform(a, beta)
```

```
            if i is 0:                  #write to habitat A list
                habA.append (r)
            else:                       #write to habitat B list
                habB.append (r)

        else:                           #d=1 (normal, corrected)...
            r = random.normalvariate(meanD, stdD)
            if r < 0:
                r = 0
            if i is 0: habA.append(r)
            else: habB.append(r)
        j = j + 1
    i = i + 1

fnum2 = file(os.path.join(path, "parameters.txt"), "w")
#Write parameter values to file:
fnum2.write("Parameters:\nSearch Cost:%s\nDefence Cost:%s\nChallenge Cost:%s\nCost\
Threshold:%s\nSample Effort:%s\nGenerations:%s\nStochastic Frequency: %s\nSeverity: %s\n" %
(sc, dc, cc, cthres, habsef, gen, eFq, eSev))

fnum2.write("Habitat Parameters:\nHabitat A: %s, %s, %s\nHabitat B: %s, %s, %s\n" % (hab1[0]\
, hab1[1], hab1[2], hab2[0], hab2[1], hab2[2]))

del fnum2
#Write group headers:
fnum.write("NULL,,,,,,,,,,,,,IFD,,,,,,,,,,,,,IDD,,,,,,,,,,,,,IPD,,,,,,,,,,,,IDD Challenger\
Counter\n")
#Write variable labels:

fnum.write("NT,NA,NB,RA,RB,COSTA,COSTB,FloatA,FloatB,MortNA,MortNB,MortFA,MortFB,NT,NA,NB,RA,RB,C
OSTA,COSTB,FloatA,FloatB,MortNA,MortNB,MortFA,MortFB,NT,NA,NB,RA,RB,COSTA,COSTB,FloatA,FloatB,Mor
tNA,MortNB,MortFA,MortFB,NT,NA,NB,RA,RB,COSTA,COSTB,FloatA,FloatB,MortNA,MortNB,MortFA,MortFB,IDD
-out\n")


#Create template cost, occupancy and sampling vectors.
cost = array([0.0] * 500)
occup = array([0] * 500)
sampler = range(500)
samplerP = range(1000)

#To keep track of floaters:
fNa = 0
fNb = 0
fFa = 0
fFb = 0
fDa = 0
fDb = 0
fPa = 0
fPb = 0

#Make the habitat lists into arrays:
habA = array(habA)
habB = array(habB)

#Save habitats for archives
pickle_file = file(os.path.join(path, "landscape.pickle"), "wb")
cPickle.dump(habA, pickle_file, 2)
cPickle.dump(habB, pickle_file, 2)
del pickle_file
```

```
##############################################################################################
#SECTION 2                                                                                   #
#Population growth definition, results definition                                           #
#Purpose: population growth by a density dependent time-lagged model: N(t+1) = Nt + r(TTL)  #
#results iterate through cost and occupancy vectors to assess fitness (incl costs) and density #
##############################################################################################
def popgrowth(hab, cost, occup, X)
#need only return n-ttl, can clear occup and cost with seed vectors
        i  = 0
        n  = 0
        r  = 0
        w  = 0
        nt = 0

        if list(occup).count(X) == 0:        #If there are no individuals in sites.
            return 0

        else:
            while i < 500:
                if occup[i] == X:            #If the site is occupied
                    w = hab[i] - cost[i]
                    if w < 0:
                        w = 0
                    r = r + w                #Calculating a total r
                    n = n + 1                #track the number in population
                i = i + 1
            #edit from previous version. Used to be r(total) + n (but all the adults die... )
            if r < 0: return 0               #Don't return negative numbers
            else: return int(r)              #same as (r(average)/n) * n

    #-----------------------------------------------------------------------------------------
    #Results calculator
    #All algorithms use the same loops, so they are written here.
    #-----------------------------------------------------------------------------------------
    def results(occupTA, occupTB, costTA, costTB, X):
        """Summarizes fitness and cost variables for entire landscape (single
strategy, based on cost, occupancy and habitat vectors."""

        RA = 0
        CA = 0
        RB = 0
        CB = 0
        j = 0
        while j < 500:
            if occupTA[j] == X:              #For every occupied site...
                RA = RA + habA[j]            #Calculate the fitness of the individual
                if habA[j] - costTA[j] < 0:
                    CA = CA + habA[j]        #THIS WAY W WILL NOT BE <0
                else:
                    CA = CA + costTA[j]

            if occupTB[j] == X:
                RB = RB + habB[j]
                if habB[j] - costTB[j] < 0:
                    CB = CB + habB[j]        #THIS WAY W WILL NOT BE <0
                else:
                    CB = CB + costTB[j]

            j = j + 1
        return(RA, CA, RB, CB)
```

```
#-----------------------------------------------------------------------------------------
#Stochastic events
#Takes strategy variables and stochastic parameters - applies stochastic mortality event.
#-----------------------------------------------------------------------------------------

def StochasticEvent(occupSA, occupSB, costSA, costSB, tFa, tFb, X, sev):
    inds = []
    MNa = 0
    MNb = 0
    MFa = 0
    MFb = 0

    for i in xrange(0, 499, 1):
        if occupSA[i] in X:
            inds.append(i)
        if occupSB[i] in X:
            inds.append(i + 500)

    randNum = len(inds) + tFa + tFb
    fat = float(sev) / 100 * randNum

    for j in xrange(0, fat, 1):
        unlucky = int(random.random() * randNum)
        if unlucky > len(inds) - 1: #A floater goes:
            fchoice = int(random.random() * 2)
            if (fchoice == 0 and tFa != 0) or tFb == 0:
                tFa = tFa - 1
                MFa = MFa + 1
            else:
                tFb = tFb - 1
                MFb = MFb + 1
        else:
            if inds[unlucky] >= 500:
                occupSB[inds[unlucky] - 500] = 0
                costSB[inds[unlucky] - 500]  = 0
                MNb = MNb + 1
            else:
                occupSA[inds[unlucky]] = 0
                costSA[inds[unlucky]]  = 0
                MNa = MNa + 1
            inds.pop(unlucky)

        #Reset the random number limit
        randNum = len(inds) + tFa + tFb

    return occupSA, occupSB, costSA, costSB, tFa, tFb, MNa, MNb, MFa, MFb

################################################################################################
    #SECTION 3                                                                                 #
    #Habitat Selection Definitions                                                             #
    #Purpose:  Definitions related to the mechanisms behind habitat and site selection.        #
################################################################################################

    #-----------------------------------------------------------------------------------------
    #3.1.1     Habitat comparision algorithm
    #Purpose:  Determines the sum of site qualities of unoccupied sites in each habitat and
    #          substracts the floaters for that
    #          habitat. Whichever habitat has a higher value is the better habitat.
    #          This is akin to (W-F) / Nv * Nv/500. The Nv (number of unoccupied sites) term
    #          cancels. 500 is the total number for each site
    #          and this is the same for both habitats so it can be safely ignored.
```

```
#-----------------------------------------------------------------------------------

def hab_comp(occupTA, occupTB, fTa, fTb):
    na    =  list(occupTA).count(0) #The number of unoccupied sites
    nb    =  list(occupTB).count(0)
    wa = 0                          #The sum of site qualities for unoccupied sites in hab a
    wb = 0
    i = 0                           #loop counter

    #Record the unoccupied sites
    while i < 500:
        if occupTA[i] == 0:
            wa = wa + habA[i]
        if occupTB[i] == 0:
            wb = wb + habB[i]
        i = i + 1

    wa = wa - fTa
    wb = wb - fTb

    #Which is better (validated for equality)
    if wa == wb:                    #if equal, choose one at random.
        i = int(random.random() * 2)
        if i == 0:
            return 'A'
        else:
            return 'B'
    elif wa > wb:
        return 'A'
    else:
        return 'B'


    #3.2
#################################################################################################
    #Definitions for the habitat selection algorithms.

    #-----------------------------------------------------------------------------------
    #3.2.1      Null model - Passive Selection
    #Purpose:  This routine is called when passive individuals choose a habitat and site. It
    #          takes
    #          the number of individual searching for sites(nt). Returns modified cost and
    #          occupancy vectors.
    #-----------------------------------------------------------------------------------

def NULL (nt, fNa, fNb):

    while nt > 0:
        sites  = []
        sitesi = []
        samps  = copy.copy(samplerP)          #To avoid sampling same sites twice
        ocost  = 0
        flagv  = 0
        habT   = concatenate((habA, habB))
        occupT = concatenate((occupNA, occupNB))
        costT  = concatenate((costNA, costNB))

        #Try/except accounts for low-cost high quality scenario (semantics)
        #when an all sites are occupied and an individual continues to sample
        try:
            while ocost < cthres:
```

```
            #Choose a site at random from those not yet sampled
            n = int (random.random() * samps.__len__())
            n = samps.pop(n)
            #Validation - no sites left
            if samps.__len__() == 0:
                flagv = 1

            if occupT[n] == 1:          #If the site is occupied...
                ocost = ocost + sc
            else:                        #If the site is empty
                #if the index is < 500, it is from habitat A.
                if habT[n] >= 1:
                    if n < 500:
                        occupNA[n] = 1
                        costNA[n]  = ocost + sc
                    else:
                        occupNB[n - 500] = 1            #Take the first empty site
                        costNB[n - 500]  = ocost + sc   #Tally the search costs
                    break
                else:
                    sites.append(occupT[n])
                    sitesi.append(n)
                    ocost = ocost + sc

            if (ocost > cthres and (sites.__len__() > 0)) or (flagv == 1) and\
               sites.__len__() > 0:

                sInd = sitesi[sites.index(max(sites))]
                if sInd < 500:
                    occupNA[sInd] = 1
                    costNA[sInd]  = ocost
                else:
                    occupNB[sInd - 500] = 1
                    costNB[sInd - 500]  = ocost

            #Catch the floaters
            elif ocost > cthres or samps.__len__() == 0:
                if n < 500:
                    fNa = fNa + 1
                else:
                    fNb = fNb + 1
                break

        nt = nt - 1

    except IndexError: break

    #If habitats are full, stop searching.
    if list(occupNA).count(0) == 0 and list(occupNB).count(0) == 0:
        i = 0
        while i < nt:
            n = int(random.random() * 2)
            if n == 0:
                fNa = fNa + 1
            else:
                fNb = fNb + 1
            i = i + 1
        break

return (occupNA, occupNB, costNA, costNB, fNa, fNb)

#------------------------------------------------------------------------------------------
```

```
#3.2.2     Ideal Free Habitat Selection
#Purpose:  This routine is called when ideal free individuals choose a habitat and site. It
#          takes
#          the number of individuals searching for sites, cost and occupancy vectors for each
#          habitat sampling effort and carry over costs (in case the individual nt = 1) was
#          bumped from a site by another searcher (IDD, IPD).
#          Returns modified cost and occupancy vectors.
#-----------------------------------------------------------------------------------------------
def IFD(nt, occupTA, costTA, occupTB, costTB, fFa, fFb, c=0):
    i = 0

    while i < nt:
        flagv  = 0
        sites  = []
        sitesi = []
        ocost  = c

        #Choose best habitat:
        flag = hab_comp(occupTA, occupTB, fFa, fFb)
        samp = copy.copy(sampler)                 #Remove sites previously sampled.

        #Set vectors for site selection in best habitat...
        if flag == 'A':
            habT   = habA
            occupT = occupTA
            costT  = costTA

        else:
            habT   = habB
            occupT = occupTB
            costT  = costTB

        #Search until unoccupied site found OR costs rise above threshold.
        while ocost < cthres and flagv == 0:
            #Choose a site at random from those not yet sampled
            n = int (random.random() * (samp.__len__()))
            n = samp.pop(n)
            #Validation for sampling all sites:
            if samp.__len__() == 0:
                flagv = 1

            if occupT[n] > 0:                             #If the site is occupied...
                ocost = ocost + sc
            else:                                         #If the site is empty
                if habT[n] >= 1:     #If the site is above the aspiration level,
                    occupT[n] = 2                         #Take the first empty site
                    costT[n]  = ocost + sc                #Tally the search costs
                    break
                else:               #If the site quality is below the aspiration level.
                    sites.append(occupT[n])
                    sitesi.append(n)
                    ocost = ocost + sc

            #Found some sites below aspiration level, ran out of cost
            if ((ocost > cthres or flagv == 1) and sites.__len__() > 0):
                sInd = sitesi[sites.index(max(sites))]
                occupT[sInd] = 2
                costT[sInd] = ocost
            #Catch the floaters.
            elif ocost > cthres or flagv == 1:
                if flag == 'A':
                    fFa = fFa + 1
```

```
                else:
                    fFb = fFb + 1
            elif samp.__len__() == 0:
                if flag == 'A':
                    fFa = fFa + 1
                else: fFb = fFb + 1

                break
        i = i + 1
    return(occupTA, occupTB, costTA, costTB, fFa, fFb)


#-----------------------------------------------------------------------------------------------
#3.2.3     Ideal Despotic Habitat Selection
#Purpose:  This routine is called when ideal despotic individuals choose a habitat and site.
#          It takes
#          the number of individuals searching for sites (nt), the cost and occupancy
#          vectors, and
#          modified sampling effort and carry-over costs in the event the individual has been
#          displaced from a previously chosen site (IDD).
#          Returns modified cost and occupancy vectors.
#-----------------------------------------------------------------------------------------------

def IDD (nt, occupTA, costTA, occupTB, costTB, fTa, fTb, habsef=habsef, c=0, IDDc=0):
    #Validate for population overshoot:
    #Ideal despotic habitat selection
    i = 0

    while i < nt:           #Loop ea. ind. in pop.
        #Reset variables
        ocost  = c          #c is legacy cost if the individual is displaced after selection.
        scnt   = 0
        flag   = hab_comp(occupTA, occupTB, fTa, fTb)
        samp   = copy.copy(sampler)                    #Marks sites that have been sampled
        sites  = []                                    #Holds a list of sites sampled.
        sitesi = []
        habs   = habsef
        IFDc   = 0                                     #Counters for challenger losers
        IPDc   = 0
        flagv  = 0

    #Sample sites and select best of:
        if flag == 'A':                                 #Choose a site from habitat A
            habT   = habA
            occupT = occupTA
            costT  = costTA
        else:
            habT   = habB
            occupT = occupTB
            costT  = costTB

        #Loop until the cost has exceed the threshold (see break exception)
         while ocost < cthres and flagv == 0:
             n = samp.pop(int(random.random() * samp.__len__()))     #Choose random site
            if samp.__len__() == 0:
                flagv = 1

            if occupT[n] == 3:                #If the site is occupied IDD...
                ocost = ocost + sc            #incur cost of sampling occupied site and
                #Attainable are added to the list.
                #If the searcher has less cost than the resident... *
```

```
  if (ocost + cc) < (costT[n]):
        sites.append(habT[n]-cc)        #Record the site. Consider part of habsef
        sitesi.append(n)
    scnt = scnt + 1
    costT[n] = costT[n] + dc            #resident incurs defense cost.
else:                                   #If the site is occupied or unoccupied ...
    ocost = ocost + sc                  #incur cost of sampling unoccupied site.
    sites.append(habT[n])               #Record site quality and index.
    sitesi.append(n)
    scnt = scnt + 1

#If the correct number of sites have been sampled, see if any are good enough:
if scnt >= habs and sites.__len__() > 0:

    #Correct for changing sampling costs:
    flag2 = 1
    while flag2 == 1:
        if sites.__len__() > 0:
            sInd = sites.index(max(sites))
            sInd2 = sitesi[sites.index(max(sites))]
            if (occupT[sInd2] == 3 and (ocost + cc) > (costT[sInd2])) \
            and sites.__len__() > 0:
                    sites.pop(sInd)
                    sitesi.pop(sInd)
            else:
                flag2 = 0
        else:
            flag2 = 2

    #If the aspiration level has been met.
    if sites.__len__() > 0 and max(sites) > 1:
        sInd = sitesi[sites.index(max(sites))]
        #if the site is unoccupied:
        if occupT[sInd] == 0:
            occupT[sInd] = 3
            costT[sInd]  = ocost
            break
        #if the site is occupied by an IFD or IPD individual:
        else:
            tcost = costT[sInd]
            tstr = occupT[sInd]
            occupT[sInd] = 3
            #IDD pays a cost for kicking individuals out
            costT[sInd] = (ocost + cc)
            if tcost < cthres:
                if tstr == 2:
                    occupTA, occupTB, costTA, costTB, fTa, fTb = IFD(1, \
                    occupTA, costTA, occupTB, costTB, fTa, fTb, tcost)
                    IFDc = IFDc + 1
                elif tstr == 4:
                    occupTA, occupTB, costTA, costTB, fTa, fTb = IPD(1, \
                    occupTA, costTA, occupTB, costTB, fTa, fTb, 1, tcost)
                    IPDc = IPDc + 1
                else:
                    occupTA, occupTB, costTA, costTB, fTa, fTb, IFDc, IDDc, \
                    IPDc = IDD(1, occupTA, costTA, occupTB, costTB, fTa, \
                    fTb, 1, tcost, IDDc)
                    IDDc = IDDc + 1
            else:
                if flag == 'A':
                    fTa = fTa + 1
                else:
```

```
                    fTb = fTb + 1
                break

#.................................................................................
            #Now need to check if costs were too high.
            #If individual did not find site better than habitat mean before costs
            #got too high...
            #or if there are no sites left to sample.
            if (scnt >= habs and ocost > cthres) or flagv == 1:
                if sites.__len__() > 0:    #If the individual found any unoccupied sites...

                    #Correct for changing sampling costs:
                    flag2 = 1
                    while flag2 == 1:
                        if sites.__len__() > 0:
                            sInd = sites.index(max(sites))
                            sInd2 = sitesi[sites.index(max(sites))]
                            if (occupT[sInd2] == 3 and (ocost + cc) > (costT[sInd2])) \
                            and sites.__len__() > 0:
                                    sites.pop(sInd)
                                    sitesi.pop(sInd)
                            else:
                                flag2 = 0
                        else:
                            flag2 = 2

            if (scnt >= habs and ocost > cthres) or flagv == 1:
                #Check again if there are available sites before proceeding with selection
                if sites.__len__() > 0:      #If the individual found any unoccupied sites...

                    sInd = sitesi[sites.index(max(sites))]
                    #if the site is unoccupied:
                    if occupT[sInd] == 0:
                        occupT[sInd] = 3
                        costT[sInd]  = ocost


                    #if the site is occupied by a IFD or IPD individual:
                    else:
                        tcost = costT[sInd]
                        tstr = occupT[sInd]
                        occupT[sInd] = 3
                        costT[sInd] = (ocost + cc)
                        if tcost < cthres:
                            if tstr == 2:
                                occupTA, occupTB, costTA, costTB, fTa, fTb = IFD(1, \
                                occupTA, costTA, occupTB, costTB, fTa, fTb, tcost)
                                IFDc = IFDc + 1
                            elif tstr == 4:
                                occupTA, occupTB, costTA, costTB, fTa, fTb = IPD(1, \
                                occupTA, costTA, occupTB, costTB, fTa, fTb, 1, tcost)
                                IPDc = IPDc + 1
                            else:
                                occupTA, occupTB, costTA, costTB, fTa, fTb, IFDc, IDDc, \
                                IPDc = IDD(1, occupTA, costTA, occupTB, costTB, fTa, fTb, 1,\
                                tcost, IDDc)
                                IDDc = IDDc + 1

                        else:
                            if flag == 'A':
                                fTa = fTa + 1
                            else:
```

```
                        fTb = fTb + 1

                #If any didn't find sites (floaters):
                else:
                    if flag == 'A':
                        fTa = fTa + 1
                    else:
                        fTb = fTb + 1

        i = i + 1

    return occupTA, occupTB, costTA, costTB, fTa, fTb, IFDc, IDDc, IPDc

#-----------------------------------------------------------------------------------------------
#3.2.4     Ideal Pre-emptive Habitat Selection
#Purpose:  This routine is called when ideal pre-emptive individuals choose a habitat and
#          site. It takes the number of individual searching for sites. Returns modified cost
#          and occupancy vectors. IPD individuals do no remove other IDD individuals or other
#          IPD individuals, but can depose IFD individuals.
#-----------------------------------------------------------------------------------------------
def IPD (nt, occupTA, costTA, occupTB, costTB, fPa, fPb, habsef=habsef,  c=0):
    i = 0

    while i < nt:
        flagv = 0
        ocost  = c                          #Reset variables
        scnt   = 0
        tcnt   = 0
        samp   = copy.copy(samplerP)        #Marks sites that have been sampled
        sites  = []                         #Holds a list of sites sampled.
        sitesi = []
        habs   = habsef

    #Sample sites and select best of
    #IPD take best of a sample of sites, regardless of habitat, so
    #treat entire landscape like a single habitat.
        habT   = concatenate((habA, habB))
        occupT = concatenate((occupTA, occupTB))
        costT  = concatenate((costTA, costTB))


        #Loop until the cost has exceed the threshold (see break exception)
        while ocost < cthres and flagv == 0:
            #Landscape # of patches.
            n = int(random.random() * (samp.__len__()))
            n = samp.pop(n)
            #Validation flag for all sites sampled.
            if samp.__len__() == 0:
                flagv = 1

            if occupT[n] > 0:                   #If the site is occupied ...
                ocost = ocost + sc              #incur cost of sampling occupied site and
                scnt = scnt + 1
            else:                        #If the site is unoccupied or occupied by IFD...
                ocost = ocost + sc              #incur cost of sampling unoccupied site.
                sites.append(habT[n])           #Record site quality and index.
                sitesi.append(n)
                scnt = scnt + 1

            #If the correct number of sites have been sampled, see if any are good enough:
            if scnt >= habs and sites.__len__() > 0:
                if max(sites) > 1:
```

```
                    sInd = sitesi[sites.index(max(sites))]
                    if sInd <= 499:                      #Hab A
                        #The site is vacant
                        if occupT[sInd] == 0:
                            occupTA[sInd] = 4
                            costTA[sInd]  = (ocost)
                        break
                    else:                                 #Hab B
                        #VACANT SITE
                        if occupT[sInd] == 0:
                            occupTB[sInd - 500] = 4
                            costTB[sInd - 500]  = (ocost)
                        break

        #Now need to check if costs were too high...
        #If individual did not find site better than habitat mean before costs got too high...
            if ((scnt >= habs and ocost > cthres or samp.__len__() == 0 )) or flagv == 1:
                if sites.__len__() > 0:
                    sInd = sitesi[sites.index(max(sites))]
                    if sInd <= 499:
                        occupTA[sInd] = 4
                        costTA[sInd] = (ocost)
                    else:
                        occupTB[sInd - 500] = 4
                        costTB[sInd - 500] = (ocost)
            #Catch the floaters
            elif ocost > cthres:
                if n < 500:
                    fPa = fPa + 1
                else:
                    fPb = fPb + 1
            elif samp.__len__() == 0:
                if n < 500:
                    fPa = fPa + 1
                else:
                    fPb = fPb + 1
         #Next individual
         i = i + 1

        return occupTA, occupTB, costTA, costTB, fPa, fPb

###################################################################################################
    #SECTION 4                                                                                    #
    #Normal Population Growth                                                                     #
    #Note: Single strategy in the landscape is growing for generations specified at               #
    #       beginning of code.                                                                    #
###################################################################################################

    invas = [0,0,0]                    #Population size achived, for use in invasion analysis.
    g = 0
    while g < gen:
        event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.

        if event == 0: sev = int(random.random() * (eSev + 1))

        #--------------------------------------------------------------------------------------
        #4.1 NULL Model population growth
        #--------------------------------------------------------------------------------------
        #Null model - passive dispersals

        #1. Allow the population to grow:
        if g > 0:
```

```
    nt1 = popgrowth(habA, costNA, occupNA, 1) - fNa
    nt2 = popgrowth(habB, costNB, occupNB, 1) - fNb
    if nt1 < 0: nt1 = 0
    if nt2 < 0: nt2 = 0
    nt  = nt1 + nt2
#set starting population here:
else:
    nt = 10

RA  = 0                    #For writing out value of r for each habitat
RB  = 0
CA  = 0
CB  = 0
fNa = 0
fNb = 0
MNNa = 0                    #Mortality counters
MNNb = 0
MNFa = 0
MNFb = 0
fnum.write('%s' % (nt))

occupNA = occup.copy()    #Reset occupancy & cost vectors
occupNB = occup.copy()
costNA  = cost.copy()
costNB  = cost.copy()

#Send empty occupancy and cost vectors with population size to habitat
#selection algorithm.
if nt > 0: occupNA, occupNB, costNA, costNB, fNa, fNb = NULL(nt, fNa, fNb)

if event == 0: #If this is a year for stochastic event. 1 in eFq chance.
    occupNA, occupNB, costNA, costNB, fNa, fNb, MNNa, MNNb, MNFa, MNFb \
    = StochasticEvent(occupNA, occupNB, costNA, costNB, fNa, fNb, [1], sev)

#Results:
RA, CA, RB, CB = results(occupNA, occupNB, costNA, costNB, 1)
NA = occupNA.sum()
NB = occupNB.sum()

#Write out results
fnum.write(",%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s," % (NA, NB, RA, RB, CA, CB, fNa, \
fNb, MNNa, MNNb, MNFa, MNFb))

#------------------------------------------------------------------------------------------
#4.2 Ideal free growth
#------------------------------------------------------------------------------------------
if g > 0:
    nt1     = popgrowth(habA, costFA, occupFA, 2) - fFa
    nt2     = popgrowth(habB, costFB, occupFB, 2) - fFb
    if nt1 < 0: nt1 = 0
    if nt2 < 0: nt2 = 0
    nt      = nt1 + nt2
#Set starting population size here.
else:
    nt = 10

RA      = 0                                #For recording output
RB      = 0
CA      = 0
CB      = 0
fFa     = 0
fFb     = 0
```

```
MFNa      = 0
MFNb      = 0
MFFa      = 0
MFFb      = 0
invas[0] = nt

occupFA  = occup.copy()             #Reset vectors
occupFB  = occup.copy()
costFA   = cost.copy()
costFB   = cost.copy()
if nt > 0: occupFA, occupFB, costFA, costFB, fFa, fFb = IFD(nt, occupFA, costFA, \
occupFB, costFB, fFa, fFb)

if event == 0: #If this is a year for stochastic event. 1 in eFq chance.
    occupFA, occupFB, costFA, costFB, fFa, fFb, MFNa, MFNb, MFFa, MFFb = \
    StochasticEvent(occupFA, occupFB, costFA, costFB, fFa, fFb, [2], sev)

#Results:
RA, CA, RB, CB = results(occupFA, occupFB, costFA, costFB, 2)
NA = (occupFA.sum() / 2)
NB = (occupFB.sum() / 2)
#write results to file
fnum.write("%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s," % (nt, NA, NB, RA, RB, CA, CB, \
fFa, fFb, MFNa, MFNb, MFFa, MFFb))

#------------------------------------------------------------------------------------------
#4.3 Ideal despotic growth
#------------------------------------------------------------------------------------------
if g > 0:
    nt1 = popgrowth(habA, costDA, occupDA, 3) - fDa
    nt2 = popgrowth(habB, costDB, occupDB, 3) - fDb
    if nt1 < 0: nt1 = 0
    if nt2 < 0: nt2 = 0
    nt   = nt1 + nt2
    invas[1] = nt
#Set stating population size here:
else:
    nt = 10

RA      = 0
RB      = 0
CA      = 0
CB      = 0
fDa     = 0
fDb     = 0
MDNa    = 0
MDNb    = 0
MDFa    = 0
MDFb    = 0
occupDA = occup.copy()
occupDB = occup.copy()
costDA  = cost.copy()
costDB  = cost.copy()

#IFDc, IDDc, and IPDc are counters for individuals kicked out of their sites
#by ID individuals
#IDDc is written at the end of the IPD data (to preserve analysis code already written).
#IFDc & IPDc are used only for the invasion analysis.
if nt > 0: occupDA, occupDB, costDA, costDB, fDa, fDb, IFDc, IDDc, IPDc = IDD(nt,\
occupDA, costDA, occupDB, costDB, fDa, fDb)

if event == 0: #If this is a year for stochastic event. 1 in eFq chance.
```

```
        occupDA, occupDB, costDA, costDB, fDa, fDb, MDNa, MDNb, MDFa, MDFb \
        = StochasticEvent(occupDA, occupDB, costDA, costDB, fDa, fDb, [3], sev)

    #Results
    RA, CA, RB, CB = results(occupDA, occupDB, costDA, costDB, 3)
    NA = (occupDA.sum() / 3)
    NB = (occupDB.sum() / 3)
    #Write results
    fnum.write("%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s," % (nt, NA, NB, RA, RB, CA, CB, \
    fDa, fDb, MDNa, MDNb, MDFa, MDFb))

    #-----------------------------------------------------------------------------------
    #4.4 Ideal pre-emptive habitat selectors.
    #-----------------------------------------------------------------------------------
    if g > 0:
        nt1 = popgrowth(habA, costPA, occupPA, 4) - fPa
        nt2 = popgrowth(habB, costPB, occupPB, 4) - fPb
        if nt1 < 0: nt1 = 0
        if nt2 < 0: nt2 = 0
        nt  = nt1 + nt2

    else:
        nt = 10

    invas[2] = nt
    RA   = 0
    RB   = 0
    CA   = 0
    CB   = 0
    fPa  = 0
    fPb  = 0
    MPNa = 0
    MPNb = 0
    MPFa = 0
    MPFb = 0

    occupPA = occup.copy()
    occupPB = occup.copy()
    costPA  = cost.copy()
    costPB  = cost.copy()

    if nt > 0: occupPA, occupPB, costPA, costPB, fPa, fPb = IPD(nt, occupPA, \
    costPA, occupPB, costPB, fPa, fPb)

    if event == 0: #If this is a year for stochastic event. 1 in eFq chance.
        occupPA, occupPB, costPA, costPB, fPa, fPb, MPNa, MPNb, MPFa, \
        MPFb = StochasticEvent(occupPA, occupPB, costPA, costPB, fPa, fPb, [4], sev)


    #Results
    RA, CA, RB, CB = results(occupPA, occupPB, costPA, costPB, 4)
    NA = (occupPA.sum() / 4)
    NB = (occupPB.sum() / 4)

    #Write results
    fnum.write("%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n" % (nt, NA, NB, RA, \
    RB, CA, CB, fPa, fPb, MPNa, MPNb, MPFa, MPFb, IDDc))

g = g + 1
#delete file object
fnum.close()
del fnum
```

```
#-------------------------------------------------------------------------------------------
#4.2 Results Summary
#Purpose: Print mean fitness values from population growth. Also records which value
#is highest: used in invasion analysis.
#-------------------------------------------------------------------------------------------
g = 0
fits = []
fnumt = file(os.path.join(path, "growth.csv"), "r")

wNULL = []
wIFD   = []
wIDD   = []
wIPD  = []

for i in 1, 2:
    fnumt.readline()

temp = fnumt.readline()
while temp:
    temp2 = temp.split(',')
    wNULL.append((float(temp2[3]) + float(temp2[4]) - float(temp2[5]) - float(temp2[6]) -\
    float(temp2[7]) - float(temp2[8])) / (float(temp2[1]) + float(temp2[2]) + \
    float(temp2[7]) + float(temp2[8])))

    wIFD.append((float(temp2[16]) + float(temp2[17]) - float(temp2[18]) - float(temp2[19])\
     - float(temp2[20]) - float(temp2[21])) / (float(temp2[14]) + float(temp2[15]) +\
       float(temp2[20]) + float(temp2[21])))

    wIDD.append((float(temp2[29]) + float(temp2[30]) - float(temp2[31]) - float(temp2[32]) \
     - float(temp2[33]) - float(temp2[34])) / (float(temp2[27]) + float(temp2[28]) + \
    float(temp2[33]) + float(temp2[34])))

    wIPD.append((float(temp2[42]) + float(temp2[43]) - float(temp2[44]) - float(temp2[45]) \
    - float(temp2[46]) - float(temp2[47])) / (float(temp2[40]) + float(temp2[41]) + \
    float(temp2[46]) + float(temp2[47])))

    temp = fnumt.readline()

#Convert to arrays
wNULL = array(wNULL)
wIFD  = array(wIFD)
wIDD  = array(wIDD)
wIPD  = array(wIPD)

while g < gen:                 #Replace 0s with exceptionally low values for gmean calc.
    if wNULL[g] <= 0:
        wNULL[g] = 0.0001
    if wIFD[g]  <= 0:
        wIFD[g]  = 0.0001
    if wIDD[g]  <= 0:
        wIDD[g]  = 0.0001
    if wIPD[g]  <= 0:
        wIPD[g]  = 0.0001

    wNULL[g] = math.log(wNULL[g], e)
    wIFD[g] = math.log(wIFD[g], e)
    wIDD[g] = math.log(wIDD[g], e)
    wIPD[g] = math.log(wIPD[g], e)

    g = g + 1
w0 = exp(mean(wNULL))
```

```
    w1 = exp(mean(wIFD))
    w2 = exp(mean(wIDD))
    w3 = exp(mean(wIPD))
    fits = [w1, w2, w3]
    flagI = fits.index(max(fits))    #Flag marks best strategy for invasion analysis.
    fnumt.close()
    del fnumt
    print """SUMMARY (Geometric mean fitnesses):\nNULL\t%s\nIFD\t%s\nIDD\t%s\nIPD\t%s""" % \
    (w0, w1, w2, w3)
    fnum = file(os.path.join(path, 'parameters.txt'), 'a')
    fnum.write("""SUMMARY (Geometric mean fitnesses):\nNULL\t%s\nIFD\t%s\nIDD\t%s\nIPD\t%s""" \
    % (w0, w1, w2, w3))
    del fnum, w0, w1, w2, w3, wNULL, wIFD, wIDD, wIPD


################################################################################################
    #SECTION 5                                                                                  #
    #Controlled density performance                                                             #
    #Note: This section should determine what the best possible strategy would be, and then     #
    #further determine                                                                          #
    #which of the strategies has the best fit to that perfect strategy                          #
################################################################################################

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    #5.1: Determine the [cost-free] distribution that maximizes fitness
    #Searches for maximum per-capita growth rate in each habitat.
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    fnum = file(os.path.join(path, "fixed_density.csv"), 'w')
    fnum.write("iteration,strategy,NA,RA,CA,NB,RB,CB,FloatA,FloatB,IDD-out\n")
    #this section runs the longest - so assessing fitness at populations sizes from 1-1000 \
    #at every 10.

    for i in xrange(10,1001,10):
        j     = 1
        acnt  = 0
        bcnt  = 0
        RA    = 0
        RB    = 0

        #convert habitat arrays back to lists, for different coding properties.
        habTA = list(habA)
        habTB = list(habB)

        #Want to know the per-capita fitness in each habitat:
        #Validate for no empty
        while j <= i:

            #if/elif - if one site has no space left, just take sites from the other \
            #site in order
            if len(habTA) == 0:
                RB = RB + habTB.pop(habTB.index(max(habTB)))
                bcnt = bcnt + 1

            elif len(habTB) == 0:
                RA = RA + habTA.pop(habTA.index(max(habTA)))
                acnt = acnt + 1

            #There are still sites left in both habitats
            else:
                #If per-capita fitness is higher in A:
                if (RA + max(habTA)) / (acnt + 1) > (RB + max(habTB)) / (bcnt + 1):
```

```
                RA = RA + habTA.pop(habTA.index(max(habTA)))
                acnt = acnt + 1

            #If per-capita fitness is higher in B:
            elif (RA + max(habTA)) / (acnt + 1) < (RB + max(habTB)) / (bcnt + 1):
                RB = RB + habTB.pop(habTB.index(max(habTB)))
                bcnt = bcnt + 1

            #If the per-capita fitness is equal in each habitat...
            else:
                #Check if there is a difference in the sites
                if max(habTA) > max(habTB):
                    RA = RA + habTA.pop(habTA.index(max(habTA)))
                    acnt = acnt + 1

                elif max(habTA) < max(habTB):
                    RB = RB + habTB.pop(habTB.index(max(habTB)))
                    bcnt = bcnt + 1

                #chose one at random
                else:
                    n = int(random.random() * 2)

                    if n == 0:
                        RA = RA + habTA.pop(habTA.index(max(habTA)))
                        acnt = acnt + 1

                    else:
                        RB = RB + habTB.pop(habTB.index(max(habTB)))
                        bcnt = bcnt + 1
        j = j + 1
    fnum.write("%s,fmax,%s,%s,,%s,%s\n" % ((i), acnt, RA, bcnt, RB))


#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    #5.2 - The performance of the NULL model at controlled densities:
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    RA   = 0                        #For writing out value of r for each habitat
    RB   = 0
    CA   = 0
    CB   = 0
    acnt = 0
    bcnt = 0
    for s in range (1,10):          #(Averaging resutls)
        fNa = 0
        fNb = 0

        occupNA = occup.copy()      #Reset occupancy & cost vectors
        occupNB = occup.copy()
        costNA  = cost.copy()
        costNB  = cost.copy()

        occupNA, occupNB, costNA, costNB, fNa, fNb = NULL (i, fNa, fNb)

        #Results
        RA, CA, RB, CB = results(occupNA, occupNB, costNA, costNB, 1)

        #WRITE RESULTS
        fnum.write("%s,NULL,%s,%s,%s,%s,%s,%s,%s,%s\n" % (i, (occupNA.sum()), RA, \
        CA, occupNB.sum(), RB, CB, fNa, fNb) )
```

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        #5.3 - The performance of the IFD model at controlled densities:
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        RA   = 0                              #For recording output
        RB   = 0
        CA   = 0
        CB   = 0
        acnt = 0
        bcnt = 0

        for s in range (1,10):
            fFa     = 0
            fFb     = 0
            nt      = i
            occupFA = occup.copy()              #Reset vectors
            occupFB = occup.copy()
            costFA  = cost.copy()
            costFB  = cost.copy()

            occupFA, occupFB, costFA, costFB, fFa, fFb = IFD(nt, occupFA, costFA, \
            occupFB, costFB, fFa, fFb)

            #results
            RA, CA, RB, CB = results(occupFA, occupFB, costFA, costFB, 2)

            #write results to file
            fnum.write("%s,IFD,%s,%s,%s,%s,%s,%s,%s,%s\n" % (i,(occupFA.sum() / 2), RA, \
            CA, (occupFB.sum() / 2), RB, CB, fFa, fFb))


#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        #5.4 - The performance of the IDD model at controlled densities:
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        RA   = 0                                #For recording output
        RB   = 0
        CA   = 0
        CB   = 0
        acnt = 0
        bcnt = 0

        for s in range (1,10):
            fDa     = 0
            fDb     = 0
            nt      = i
            occupDA = occup.copy()              #Reset vectors
            occupDB = occup.copy()
            costDA  = cost.copy()
            costDB  = cost.copy()

            #IFDc and IPDc are used only in the invasion analysis.
            occupDA, occupDB, costDA, costDB, fDa, fDb, IFDc, IDDc, IPDc = \
            IDD(nt, occupDA, costDA, occupDB, costDB, fDa, fDb)

            #Results
            RA, CA, RB, CB = results(occupDA, occupDB, costDA, costDB, 3)
            #write results
            fnum.write("%s,IDD,%s,%s,%s,%s,%s,%s,%s,%s,%s\n" % (i, (occupDA.sum() / 3), \
            RA, CA, (occupDB.sum() / 3), RB, CB, fDa, fDb, IDDc))


#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
        #5.5 - The performance of the IPD model at controlled densities:
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        RA   = 0                              #For recording output
        RB   = 0
        CA   = 0
        CB   = 0
        acnt = 0
        bcnt = 0

        for s in range (1,10):
            fPa     = 0
            fPb     = 0
            nt      = i
            occupPA = occup.copy()              #Reset vectors
            occupPB = occup.copy()
            costPA  = cost.copy()
            costPB  = cost.copy()

            occupPA, occupPB, costPA, costPB, fPa, fPb = IPD(nt, occupPA, costPA, \
            occupPB, costPB, fPa, fPb)

            #Results
            RA, CA, RB, CB = results(occupPA, occupPB, costPA, costPB, 4)
            #write results
            fnum.write("%s,IPD,%s,%s,%s,%s,%s,%s,%s,%s\n" % (i, (occupPA.sum() / 4), \
            RA, CA, (occupPB.sum() / 4), RB, CB, fPa ,fPb))

        #Section counter - increase density
        i = i + 1

    del fnum


################################################################################################
    #SECTION 6                                                                                 #
    #Invasion Analysis (modified from Ranta and Kaitala 1999)                                  #
    #SECTION IN PROGRESS - CURRENTLY NOT WORKING - Oct 2009                                    #
    #       Considers the population size achieved after the growth phase above as the         #
    #       establishment phase.                                                               #
    #       Population size is recorded from last generation. Then competing strategies are    #
    #       added at very low density. After a thousand more generations the results are sampled #
    #       for 100 generations. This whole process is repeated several times. Ideally a       #
    #       bifurcation diagram could be created from the data, showing                        #
    #       success of invading strategy across a range of a model parameter.                  #
    #       Uses summary information from above. Takes the best strategy, and assesses stability #
    #       to invasion from other strategies (each independently, and together).              #
    #       The passive selection strategy is excluded from this analysis                      #
################################################################################################


#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#     #6.1    IFD IS THE BEST STRATEGY                                                         +
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#     fnum = file(os.path.join(path, "invasion.csv"), 'w')
#     fnum.write("Trial,StochasticSeverity,Resident,NA,NB,1stInvader,NA,NB,2ndInvader,\
#     NA,NB,IFDc,IDDc,IPDc\n")
#
#     #If the best strategy has crashed, start it off with a low population...
#     if invas[flagI] == 0: invas[flagI] = 10
#
#     for x in xrange(1,11,1):     #run analysis 10x, in case one strategy crashes...
#
```

```
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#    #6.1   IFD IS THE BEST STRATEGY                                                            +
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#        #If IFD is best:
#        if flagI == 0:
#
##***********************************************************************************************
    #6.1.1 Introduce IDD
#************************************************************************************************
#            NTF = invas[0]
#            f1 = 0
#            NTD = 1
#            i = 0
#            while i < 1100:
#                occupIA = occup.copy()
#                occupIB = occup.copy()
#                costIA  = cost.copy()
#                costIB  = cost.copy()
#                fTa = 0
#                fTb = 0
#
#                while NTF > 0 or NTD > 0:
#                    #validation for one pop having all chosen sites:
#                    if NTF == 0:
#                        occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                        IDD(NTD, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                        break
#                    elif NTD == 0:
#                        occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(NTF, \
#                        occupIA, costIA, occupIB, costIB, fTa, fTb)
#                        break
#                    #f1/f2 flags used to choose a strategy at random, and
#                    #chose the second strategy the next time by.
#                    if f1 == 0:
#                        f2 = int(random.random() * 2)
#                        f1 == 1
#                    else:
#                        if f2 == 1:
#                            f2 = 0
#                        else:
#                            f2 = 1
#                        f1 == 0
#
#                    #IFD first
#                    if f2 == 0:
#
#                        #Allow one IFD individual to choose habitat & site
#                        occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(1, occupIA, \
#                        costIA, occupIB, costIB, fTa, fTb)
#                        NTF = NTF - 1
#
#                    #IDD
#                   if f2 == 1:
#
#                        #Allow one IDD indiidual to choose habitat & site
#                        occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, \
#                        IPDc = IDD(1, occupIA, costIA, occupIB, costIB, fTa, fTb,IDDc=IDDc)
#                        NTD = NTD - 1
#
#               #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
```

```
#               if event == 0:
#                   sev = int(random.normalvariate(40,10))
#                    occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, \
#                    b4 = StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, \
#                    [2,3,4], sev)
#                else: sev = 0
#               #Proportionate distribution of floater effect
#                nta  = popgrowth(habA, costIA, occupIA, 2)
#                ntb  = popgrowth(habB, costIB, occupIB, 2)
#                ntaa = popgrowth(habA, costIA, occupIA, 3)
#                ntbb = popgrowth(habB, costIB, occupIB, 3)
#
#                #Correct for floaters.
#                ta = nta + ntaa
#                tb = ntb + ntbb
#                if ta > 0:
#                    nta  = nta - round((float(nta) / (ta)) * fTa)
#                    ntaa = ntaa - round((float(ntaa) / (ta)) * fTa)
#                if tb > 0:
#                    ntb  = ntb - round((float(ntb) / (tb)) * fTb)
                   ntbb = ntbb - round((float(ntbb) / (tb)) * fTb)

#                #Corrections:
#                if nta  < 0: nta  = 0
#                if ntb  < 0: ntb  = 0
#                if ntaa < 0: ntaa = 0
#                if ntbb < 0: ntbb = 0

#                NTD = ntaa + ntbb
#                NTF = nta + ntb

#               if i > 99:
#                   fnum.write("%s, %s, IFD, %s, %s, IDD, %s, %s,,,,%s,%s\n" % \
#                  (x, sev, nta, ntb, ntaa, ntbb, IFDc, IDDc))

#              #This validation truncates the analysis if one population crashes to extinction.
#              if NTF == 0:
#                  break
#              if NTD == 0:
                   break

#                i = i + 1

#******************************************************************************************
    #6.1.2 Introduce IPD
#******************************************************************************************
#           NTF = invas[0]
#           f1 = 0
#           NTP = 1
#           i = 0
#           while i < 1100:
#               occupIA = occup.copy()
#               occupIB = occup.copy()
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa = 0
#               fTb = 0
#
#               while NTF > 0 or NTP > 0:
#                   #Validation for all of one pop having chosen sites.
#                   if NTF == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(NTP, occupIA, \
```

```
#                       costIA, occupIB, costIB, fTa, fTb)
#                       break
#                   elif NTP == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(NTD, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       break
#
#                   #f1/f2 flags used to choose a strategy at random, and chose
#                   #the second strategy the next time by.
#                   if f1 == 0:
#                       f2 = int(random.random() * 2)
#                       f1 == 1
#                   else:
#                       if f2 == 1:
#                           f2 = 0
#                       else:
#                           f2 = 1
#                       f1 == 0
#
#                   #IFD
#                   if f2 == 0:
#                       #Allow one IFD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTF = NTF - 1
#
#                   #IPD
#                   if f2 == 1:
#                       #Allow one IDD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTP = NTP - 1
#
#               #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#               if event == 0:
#                   sev = int(random.normalvariate(40,10))
#                   occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 =\
#                   StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#               else: sev = 0
#
#               nta  = popgrowth(habA, costIA, occupIA, 2)
#               ntb  = popgrowth(habB, costIB, occupIB, 2)
#               ntaa = popgrowth(habA, costIA, occupIA, 4)
#               ntbb = popgrowth(habB, costIB, occupIB, 4)
#
#               #Correct for floaters:
#               ta = nta + ntaa
#               tb = ntb + ntbb
#               if ta > 0:
#                   nta  = nta - round((float(nta) / (ta)) * fTa)
#                   ntaa = ntaa - round((float(ntaa) / (ta)) * fTa)
#               if tb > 0:
#                   ntb  = ntb - round((float(ntb) / (tb)) * fTb)
#                   ntbb = ntbb - round((float(ntbb) / (tb)) * fTb)
#
#               #Corrections:
#               if nta  < 0: nta  = 0
#               if ntb  < 0: ntb  = 0
#               if ntaa < 0: ntaa = 0
#               if ntbb < 0: ntbb = 0
```

```
#             NTF = nta + ntb
#             NTP = ntaa + ntbb
#
#             if i > 99:
#                 fnum.write("%s,%s,IFD,%s,%s,IPD,%s,%s\n" % (x, sev, nta, ntb, ntaa, ntbb))
#
#               #truncate analysis if a strategy crashes.
#               if NTF == 0:
#                   break
#               if NTP == 0:
#                   break
#               i = i + 1


#*************************************************************************************************
    #6.1.3 Introduce IDD & IPD
#*************************************************************************************************
#           NTF = invas[0]                      #Reusing the same starting conditions
#           NTP = 0
#           NTD = 0
#           xg   = int(random.random() * 100)   #A random generation to introduce strategy
#           yg   = int(random.random() * 100)
#           f3   = []                           #List of flags
#
#           i = 0
#           while i < 1100:
#               occupIA = occup.copy()
#               occupIB = occup.copy()
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa = 0
#               fTb = 0
#
#               if i == xg:
#                   NTP = 1
#               if i == yg:
#                   NTD = 1
#
#               while NTF > 0 or NTP > 0 or NTD > 0:
#
#                   if f3 == []:
#                       f3 = [0,1,2]
#                   f2 = f3.pop(int(random.random() * (f3.__len__() - 1)))
#
#                   #IFD
#                   if f2 == 0 and NTF > 0:
#                       #Allow one IFD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(1, \
#                       occupIA, costIA, occupIB, costIB, fTa, fTb)
#                       NTF = NTF - 1
#
#                   #IPD
#                   if f2 == 1 and NTP > 0:
#                       #Allow one IDD indiidual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(1, \
#                       occupIA, costIA, occupIB, costIB, fTa, fTb)
#                       NTP = NTP - 1
#
#                   if f2 == 2 and NTD > 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = IDD(1,
#occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                       NTD = NTD - 1
```

```
#
#               #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#               if event == 0:
#                   sev = int(random.normalvariate(40,10))
#                   occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 =\
#                   StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#               else: sev = 0
#
#               nta   = popgrowth(habA, costIA, occupIA, 2)
#               ntb   = popgrowth(habB, costIB, occupIB, 2)
#               ntaa  = popgrowth(habA, costIA, occupIA, 3)
#               ntbb  = popgrowth(habB, costIB, occupIB, 3)
#               ntaaa = popgrowth(habA, costIA, occupIA, 4)
#               ntbbb = popgrowth(habB, costIB, occupIB, 4)
#
#               #Correct for fitness
#               ta = nta + ntaa + ntaaa
#               tb = ntb + ntbb + ntbbb
#               if ta > 0:
#                   nta   = nta - round((float(nta) / (ta)) * fTa)
#                   ntaa  = ntaa - round((float(ntaa) / (ta)) * fTa)
#                   ntaaa = ntaaa - round((float(ntaaa) / (ta)) * fTa)
#               if tb > 0:
#                   ntb   = ntb - round((float(ntb) / (tb)) * fTb)
#                   ntbb  = ntbb - round((float(ntbb) / (tb)) * fTb)
#                   ntbbb = ntbbb - round((float(ntbbb) / (tb)) * fTb)
#
#               #Corrections:
#               if nta   < 0: nta   = 0
#               if ntb   < 0: ntb   = 0
#               if ntaa  < 0: ntaa  = 0
#               if ntbb  < 0: ntbb  = 0
#               if ntaaa < 0: ntaaa = 0
#               if ntbbb < 0: ntbbb = 0
#
#               NTF = nta + ntb
#               NTD = ntaa + ntbb
#               NTP = ntaaa + ntbbb
#
#               if i > 99:
#                   fnum.write("%s,%s,IFD,%s,%s,IDD,%s,%s,IPD,%s,%s,%s,%s,%s\n" % (x, sev, \
#                   nta, ntb, ntaa, ntbb, ntaaa, ntbbb, IFDc, IDDc, IPDc))
#
#               i = i + 1


#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   #6.2   IDD IS THE BEST STRATEGY                                                               +
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#       elif flagI == 1:

#*************************************************************************************************
   #6.2.1 Introduce IFD
#*************************************************************************************************
#           NTD = invas[1]
#           f1  = 0
#           NTF = 1
#           i   = 0
#           while i < 1100:
#               occupIA = occup.copy()
#               occupIB = occup.copy()
```

```
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa  = 0
#               fTb  = 0
#
#               while NTF > 0 or NTD > 0:
#                   #validation for one pop having chosen sites
#                   if NTF == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                       IDD(NTD, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                       break
#
#                   elif NTD == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                       IFD(NTF, occupIA, costIA, occupIB, costIB, fTa, fTb)
#                       break
#
#                   #f1/f2 flags used to choose a strategy at random, and chose the
#                   #strategy the next time by.
#                   if f1 == 0:
#                       f2 = int(random.random() * 2)
#                       f1 == 1
#                   else:
#                       if f2 == 1:
#                           f2 = 0
#                       else:
#                           f2 = 1
#                       f1 == 0
#
#                   #IFD
#                   if f2 == 0:
#                       #Allow one IFD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(1, occupIA, costIA,\
#                       occupIB, costIB, fTa, fTb)
#                       NTF = NTF - 1
#
#                   #IDD
#                   if f2 == 1:
#                       #Allow one IDD indiidual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                       IDD(1, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                       NTD = NTD - 1
#
#               #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#               if event == 0:
#                   sev = int(random.normalvariate(40, 10))
#                   occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 \
#                   = StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#               else: sev = 0
#
#               nta  = popgrowth(habA, costIA, occupIA, 2)
#               ntb  = popgrowth(habB, costIB, occupIB, 2)
#               ntaa = popgrowth(habA, costIA, occupIA, 3)
#               ntbb = popgrowth(habB, costIB, occupIB, 3)
#
#               #Correction for floaters
#               ta = nta + ntaa
#               tb = ntb + ntbb
#               if ta > 0:
#                   nta  = nta - round((float(nta) / (ta)) * fTa)
#                   ntaa = ntaa - round((float(ntaa) / (ta)) * fTa)
```

```
#               if tb > 0:
#                   ntb  = ntb - round((float(ntb) / (tb)) * fTb)
#                   ntbb = ntbb - round((float(ntbb) / (tb)) * fTb)
#
#               if nta  < 0: nta  = 0
#               if ntb  < 0: ntb  = 0
#               if ntaa < 0: ntaa = 0
#               if ntbb < 0: ntbb = 0
#
#               NTF = nta + ntb
#               NTD = ntaa + ntbb
#
#               if i > 99:
#                   fnum.write("%s,%s,IDD, %s, %s, IFD, %s, %s,,,,%s,%s\n" % (x, sev, ntaa, \
#                   ntbb, nta, ntb, IFDc, IDDc))
#
#               #Truncate analysis if one population has crashed
#               if NTF == 0:
#                   break
#               elif NTD == 0:
#                   break
#               i = i + 1

#************************************************************************************************
    #6.2.2 Introduce IPD
#************************************************************************************************
#           NTD = invas[1]
#           f1 = 0
#           NTP = 1
#           i = 0
#           while i < 1100:
#               occupIA = occup.copy()
#               occupIB = occup.copy()
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa = 0
#               fTb = 0
#
#               while NTD > 0 or NTP > 0:
#                   #Validation for one pop having completely chosen sites
#                   if NTD == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(NTP, \
#                       occupIA, costIA, occupIB, costIB, fTa, fTb)
#                       break
#                   elif NTP == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                       IDD(NTD, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                       break
#
#                   #f1/f2 flags used to choose a strategy at random, and
#                   #chose the second strategy the next time by.
#                   if f1 == 0:
#                       f2 = int(random.random() * 2)
#                       f1 == 1
#                   else:
#                       if f2 == 1:
#                           f2 = 0
#                       else:
#                           f2 = 1
#                       f1 == 0
#
#                   #IDD
```

```
#                    if f2 == 0:
#                        occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                        IDD(1, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                        NTD = NTD - 1
#
#                    #IPD
#                    if f2 == 1:
#                        occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(1, occupIA, \
#                        costIA, occupIB, costIB, fTa, fTb)
#                        NTP = NTP - 1
#
#                #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#                if event == 0:
#                    sev = int(random.normalvariate(40, 10))
#                    occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 \
#                    = StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#                else: sev = 0
#
#                nta  = popgrowth(habA, costIA, occupIA, 3)
#                ntb  = popgrowth(habB, costIB, occupIB, 3)
#                ntaa = popgrowth(habA, costIA, occupIA, 4)
#                ntbb = popgrowth(habB, costIB, occupIB, 4)
#
#                #Correction for floaters:
#                ta = nta + ntaa
#                tb = ntb + ntbb
#
#                if ta > 0:
#                    nta  = nta - round((float(nta) / (ta)) * fTa)
#                    ntaa = ntaa - round((float(ntaa) / (ta)) * fTa)
#                if tb > 0:
#                    ntb  = ntb - round((float(ntb) / (tb)) * fTb)
#                    ntbb = ntbb - round((float(ntbb) / (tb)) * fTb)
#
#                if nta  < 0: nta  = 0
#                if ntb  < 0: ntb  = 0
#                if ntaa < 0: ntaa = 0
#                if ntbb < 0: ntbb = 0
#
#                NTD  = nta + ntb
#                NTP  = ntaa + ntbb
#
#                if i > 99:
#                    fnum.write("%s, %s, IDD, %s, %s, IPD, %s, %s,,,,,%s,%s\n" % (x, \
#                    sev, nta, ntb, nta, ntbb, IDDc, IPDc))
#                #Validation for one strategy being excluded
#                if NTD == 0:
#                    break
#                elif NTP == 0:
#                    break
#                i = i + 1

#*************************************************************************************************
    #6.2.3 Introduce IFD & IPD
#*************************************************************************************************
#          NTD = invas[1]              #Reusing the same starting conditions
#           NTF = 0
#           NTP = 0
#           xg  = int(random.random() * 100)
#           yg  = int(random.random() * 100)
#           f3  = [] #List of flags
```

```
#           i   = 0
#           while i < 1100:
#               occupIA = occup.copy()
#               occupIB = occup.copy()
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa = 0
#               fTb = 0
#
#               #Introduce competitors:
#               if i == xg:
#                   NTF = 1
#               if i == yg:
#                   NTP = 1
#
#               while NTF > 0 or NTP > 0 or NTD > 0:                    #who chooses habitat:
#                   if f3 == []:
#                       f3 = [0,1,2]
#                   f2 = f3.pop(int(random.random() * (f3.__len__() - 1)))
#
#
#                   #IFD
#                   if f2 == 0 and NTF > 0:
#                       #Allow one IFD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTF = NTF - 1
#                   #IPD
#
#                   if f2 == 1 and NTP > 0:
#                       #Allow one IDD indiidual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTP = NTP - 1
#
#                   if f2 == 2 and NTD > 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                       IDD(1, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                       NTD = NTD - 1
#
#               #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#               if event == 0:
#                   sev = int(random.normalvariate(40, 10))
#                   occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 =\
#                   StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#               else: sev = 0

#               nta   = popgrowth(habA, costIA, occupIA, 3)
#               ntaa  = popgrowth(habA, costIA, occupIA, 4)
#               ntaaa = popgrowth(habA, costIA, occupIA, 2)
#               ntb   = popgrowth(habB, costIB, occupIB, 3)
#               ntbb  = popgrowth(habB, costIB, occupIB, 4)
#               ntbbb = popgrowth(habB, costIB, occupIB, 2)
#
#               #Fitness corrections:
#               ta = nta + ntaa + ntaaa
#               tb = ntb + ntbb + ntbbb
#               if ta > 0:
##                   nta   = nta - round((float(nta) / (ta)) * fTa)
#                   ntaa  = ntaa - round((float(ntaa) / (ta)) * fTa)
#                   ntaaa = ntaaa - round((float(ntaaa) / (ta)) * fTa)
```

```
#               if tb > 0:
#                   ntb   = ntb - round((float(ntb) / (tb)) * fTb)
#                   ntbb  = ntbb - round((float(ntbb) / (tb)) * fTb)
#                   ntbbb = ntbbb - round((float(ntbbb) / (tb)) * fTb)

#               if nta   < 0: nta   = 0
#               if ntb   < 0: ntb   = 0
#               if ntaa  < 0: ntaa  = 0
#               if ntbb  < 0: ntbb  = 0
#               if ntaaa < 0: ntaaa = 0
#               if ntbbb < 0: ntbbb = 0

#               NTD   = nta + ntb
#               NTP   = ntaa + ntbb
#               NTF   = ntaaa + ntbbb

#               if i > 99:
#                   fnum.write("%s, %s, IDD, %s, %s, IPD, %s, %s, IFD, %s, %s, %s, %s, %s\n" \
#                   % (x, sev, nta, ntb, ntaa, ntbb, ntaaa, ntbbb, IFDc, IDDc, IPDc))

#               i = i + 1


#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    #6.3   IPD IS THE BEST STRATEGY                                                              +
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#       else:

#************************************************************************************************
    #6.3.1 Introduce IDD
#************************************************************************************************
#           NTP = invas[2]
#           f1 = 0
#           NTD = 1
#           i = 0
#           while i < 1100:
#               occupIA = occup.copy()
#               occupIB = occup.copy()
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa = 0
#               fTb = 0
#
#               while NTP > 0 or NTD > 0:
#                   #Validation for one pop going to 0
#                   if NTP == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                       IDD(NTD, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                       break
#
#                   elif NTD == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(NTP, occupIA,
#                       costIA, occupIB, costIB, fTa, fTb)
#                       break
#
#                   #f1/f2 flags used to choose a strategy at random, and chose
#                   #the second strategy the next time by.
#                   if f1 == 0:
#                       f2 = int(random.random() * 2)
#                       f1 == 1
#                   else:
#                       if f2 == 1:
```

```
#                          f2 = 0
#                      else:
#                          f2 = 1
#                      f1 == 0
#
#                  #IFD first
#                  if f2 == 0:
#                      #Allow one IFD individual to choose habitat & site
#                      occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(1, occupIA, \
#                      costIA, occupIB, costIB, fTa, fTb)
#                      NTP = NTP - 1
#
#                  #IDD
#                  if f2 == 1:
#                      Allow one IDD indiidual to choose habitat & site
#                      occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                      IDD(1, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#
#                     NTD = NTD - 1

#              #Stochastic Influence
#             event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#              if event == 0:
#                  sev = int(random.normalvariate(40, 10))
#                  occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 =\
#                  StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#              else: sev = 0

#              nta  = popgrowth(habA, costIA, occupIA, 4)
#              ntb  = popgrowth(habB, costIB, occupIB, 4)
#              ntaa = popgrowth(habA, costIA, occupIA, 3)
#              ntbb = popgrowth(habB, costIB, occupIB, 3)
#
#              #Fitness correction:
#              ta = nta + ntaa
#              tb = ntb + ntbb
#
#              if ta > 0:
#                  nta  = nta - round((float(nta) / (ta)) * fTa)
#                  ntaa = ntaa - round((float(ntaa) / (ta)) * fTa)
#              if tb > 0:
#                  ntb  = ntb - round((float(ntb) / (tb)) * fTb)
#                  ntbb = ntbb - round((float(ntbb) / (tb)) * fTb)
#
#              if nta   < 0: nta   = 0
#              if ntb   < 0: ntb   = 0
#              if ntaa  < 0: ntaa  = 0
#              if ntbb  < 0: ntbb  = 0
#
#              NTP  = nta + ntb
#              NTD  = ntaa + ntbb
#
#              if i > 99:
#                  fnum.write("%s, %s, IPD, %s, %s, IDD, %s, %s,,,,,%s,%s\n" % (x, sev, nta,\
#                  ntb, ntaa, ntbb, IDDc, IPDc))
#              #Validation for one strategy being excluded
#              if NTP == 0:
#                  break
#              elif NTD == 0:
#                  break
#
#              i = i + 1
```

```
#
#****************************************************************************************************
    #6.3.2 Introduce IFD
#****************************************************************************************************
#           NTP = invas[2]
#           f1  = 0
#           NTF = 1
#           i   = 0
#           while i < 1100:
#               occupIA = occup.copy()
#               occupIB = occup.copy()
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa = 0
#               fTb = 0
#
#               while NTF > 0 or NTP > 0:
#                   #Validation for one pop having selected its sites
#                   if NTF == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(NTP, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       break
#
#                   elif NTP == 0:
#                       occupIA, occupIB, costIA, costIB, fTa, fTb =  IFD(NTF, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       break
#
#                   #f1/f2 flags used to choose a strategy at random, and chose the
#                   #second strategy the next time by.
#                   if f1 == 0:
#                       f2 = int(random.random() * 2)
#                       f1 == 1
#                   else:
#                       if f2 == 1:
#                           f2 = 0
#                       else:
#                           f2 = 1
#                       f1 == 0
#
#                   #IFD
#                   if f2 == 0:
#                       #Allow one IFD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTF = NTF - 1
#
##                   #IPD
#                   if f2 == 1:
#                       #Allow one IDD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTP = NTP - 1
#
#
#               #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#               if event == 0:
#                   sev = int(random.normalvariate(40, 10))
#                   occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 =\
#                   StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#               else: sev = 0
```

```
#
#               nta  = popgrowth(habA, costIA, occupIA, 2)
#               ntb  = popgrowth(habB, costIB, occupIB, 2)
#               ntaa = popgrowth(habA, costIA, occupIA, 4)
#               ntbb = popgrowth(habB, costIB, occupIB, 4)
#
#               #Floater correction
#               ta = nta + ntaa
#               tb = ntb + ntbb
#
#               if ta > 0:
#                   nta  = nta - round((float(nta) / (ta)) * fTa)
#                   ntaa = ntaa - round((float(ntaa) / (ta)) * fTa)
#
#               if tb > 0:
#                   ntb  = ntb - round((float(ntb) / (tb)) * fTb)
#                   ntbb = ntbb - round((float(ntbb) / (tb)) * fTb)
#
#               if nta   < 0: nta   = 0
#               if ntb   < 0: ntb   = 0
#               if ntaa  < 0: ntaa  = 0
#               if ntbb  < 0: ntbb  = 0
#
#               NTF = nta + ntb
#               NTP = ntaa + ntbb
#
#               if i > 99:
#                   fnum.write("%s, %s, IPD, %s, %s, IFD, %s, %s\n" % (x, sev, ntaa, ntbb, \
#                   nta, ntb))
#
#               #Validation for one strategy being excluded
#               if NTF == 0:
#                   break
#               elif NTP == 0:
#                   break
#               i = i + 1
#
#
#*********************************************************************************************
   #6.3.3 Introduce IFD & IDD
#*********************************************************************************************
#           NTP = invas[2]                          #Reusing the same starting conditions
#           NTF = 0
#           NTD = 0
#           xg   = int(random.random() * 99)         #generation to introduce IFD
#           yg   = int(random.random() * 99)         #generation to introduce IDD
#           f3   = []                               #List of flags
#           i    = 0
#           while i < 1100:
#
#               occupIA = occup.copy()
#               occupIB = occup.copy()
#               costIA  = cost.copy()
#               costIB  = cost.copy()
#               fTa = 0
#               fTb = 0
#
#               #Introduce competing strategies in a randomly selected generation
#               #within the first 100.
#               if i == xg:
#                   NTF = 1
#               if i == yg:
```

```
#                   NTD = 1
#
#               while NTF > 0 or NTP > 0 or NTD > 0:     #Let all individuals choose habitats
#                   if f3 == []:
#                       f3 = [0,1,2]
#                   f2 = f3.pop(int(random.random() * (f3.__len__() - 1)))
#
#                   #IFD
#                   if f2 == 0 and NTF > 0:
#                       #Allow one IFD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IFD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTF = NTF - 1
#
#                   #IPD
#                   if f2 == 1 and NTP > 0:
#                       #Allow one IPD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb = IPD(1, occupIA, \
#                       costIA, occupIB, costIB, fTa, fTb)
#                       NTP = NTP - 1
#
#                   if f2 == 2 and NTD > 0:
#                       #Allow one IDD individual to choose habitat & site
#                       occupIA, occupIB, costIA, costIB, fTa, fTb, IFDc, IDDc, IPDc = \
#                       IDD(1, occupIA, costIA, occupIB, costIB, fTa, fTb, IDDc=IDDc)
#                       NTD = NTD - 1
#
#               #Stochastic Influence
#              event = int(random.random() * (eFq + 1)) #A random year for the stochastic event.
#              if event == 0:
#                   sev = int(random.normalvariate(40, 10))
#                   occupIA, occupIB, costIA, costIB, fTa, fTb, b1, b2, b3, b4 = \
#                   StochasticEvent(occupIA, occupIB, costIA, costIB, fTa, fTb, [2,3,4], sev)
#               else: sev = 0
#
#               nta   = popgrowth(habA, costIA, occupIA, 2)
#               ntb   = popgrowth(habB, costIB, occupIB, 2)
#               ntaa  = popgrowth(habA, costIA, occupIA, 4)
#               ntbb  = popgrowth(habB, costIB, occupIB, 4)
#               ntaaa = popgrowth(habA, costIA, occupIA, 3)
#               ntbbb = popgrowth(habB, costIB, occupIB, 3)
#
#               #Floater correction
#               ta = nta + ntaa + ntaaa
#               tb = ntb + ntbb + ntbbb
#               if ta > 0:
#                   nta   = nta - round((float(nta) / (ta)) * fTa)
#                   ntaa  = ntaa - round((float(ntaa) / (ta)) * fTa)
#                   ntaaa = ntaaa - round((float(ntaaa) / (ta)) * fTa)
#               if tb > 0:
#                   ntb   = ntb - round((float(ntb) / (tb)) * fTb)
#                   ntbb  = ntbb - round((float(ntbb) / (tb)) * fTb)
#                   ntbbb = ntbbb - round((float(ntbbb) / (tb)) * fTb)
#
#               if nta   < 0: nta   = 0
#               if ntb   < 0: ntb   = 0
#               if ntaa  < 0: ntaa  = 0
#               if ntbb  < 0: ntbb  = 0
#               if ntaaa < 0: ntaaa = 0
#               if ntbbb < 0: ntbbb = 0
#
#               NTF = nta + ntb
```

```
#               NTP = ntaa + ntbb
#               NTD = ntaaa + ntbbb
#
#               if i > 99:
#                   fnum.write("%s, %s, IPD, %s, %s, IFD, %s, %s, IDD, %s, %s, %s, %s, %s\n" \
#                   % (x, sev, ntaa, ntbb, nta, ntb, ntaaa, ntbbb, IFDc, IDDc, IPDc))
#
#               i = i + 1
#    del fnum
```