Optimal Sperm Strategies for Male Butterflies when Females are Abundant

Supervisors: Dr Max Reuter (Department of Biology, Galton Lab, UCL)
Dr Greg Hurst (Department of Biology, Galton Lab, UCL)

3,580 words

Abstract

Bacterial infections such as the male-killing *Wolbachia* can infect butterflies and cause distortions in the operational sex ratio (OSR) of the population by increasing the number of females relative to males. The excess of females leads to increased opportunities for males to mate, and alters the optimum sperm allocation strategy. Mutant males who have a strategy which fertilises more eggs than will have an immediate advantage over rival males, and such mutations would be expected to fixate rapidly leading to fast evolution of sperm strategy towards the new optimum. This essay presents a model of both female and male mating behaviour which is solved using a method of dynamic programming combined with simulation to yield the optimal sperm strategy for a given initial OSR. The results are found to correspond with the literature in predicting a rise in female promiscuity with increasing OSR until scarcity of males starves females of mating opportunities. An increase in sperm competition is shown to be the price males pay to take advantage of extra mating opportunities offered by favourable OSR, and is modelled as a consequence of sperm strategy rather than as a driving force.

Introduction

The operational sex ratio (OSR) of a population is the ratio of females to males who are available for mating. Bacterial parasites such as *Wolbachia* can infect butterfly populations (such as *Hypolimnas bolina* found in southeast Asia and the Pacific Islands) and cause distortion of the OSR by either killing male offspring or feminising males.

Assuming that the butterfly population has evolved over time to exhibit an optimal mating strategy which works for the 'usual' OSR of approximately 1:1, the sudden shift in OSR to, say, 20:1 or even, in extreme cases, 100:1, will mean that the mating strategies are no longer optimal. This can result in sperm limitation where mated females do not have enough sperm to fully fertilize all of their eggs, and a large proportion of the female population die without having a chance to mate at all [1]. Under such conditions it would be expected that any change in male mating behaviour in the direction of a more optimal strategy (that is, one which produces more offspring than rivals) will very rapidly spread through the population leading to rapid evolution of sperm strategy.

The modelling of sperm allocation strategies is a much-studied area with over 35 years of academic research invested in it: in 1970 Parker [2] suggested that male competition in mating can continue after the event since females may re-mate with rival males leading to direct competition at the sperm level over egg fertilization. Since then much effort has been expended identifying sperm competition in various

species (especially insects [3]) and studying the impact of different mating systems on sperm allocation strategies [4,5,6] as well how different models of the degree of sperm competition (e.g. 'intensity' vs. 'risk' models) affect sperm allocation strategy [7,8,9,10].

Sperm strategies are strongly dependent on the mating system. In the case of butterflies, males deliver sperm within a spermatophore capsule. Females extract and store the sperm in a sperm storage organ for later use in fertilizing eggs, before re-mating with another male. In this way a female can collect the sperm from more than one male, giving rise to sperm competition. The process by which the female selects which sperm to use when fertilizing her eggs is not precisely known. Parker [11] describes the possibility of so-called sperm raffles where the choice of sperm to use can be random but 'fair' (chance of reproductive success is proportional to the sperm investment relative to the other males), or 'loaded' where the first of last male displaces some of his rivals sperm, or the female actively discounts the value of some of the sperm within the 'raffle'.

The use of spermatophores to deliver sperm is a useful feature of butterfly physiology because the spermatophore capsule remains in the female reproductive tract until death and the size reflects the amount of sperm it contained. Therefore by extracting and measuring spermatophores from mated females it is possible to reconstruct the female mating history, and make inferences about the male strategy being played. Such practical experiments are necessary to provide a physical basis for the various parameters included in sperm models [1,4].

This essay is primarily concerned with exploring how the optimum sperm allocation strategy for butterflies (e.g. *H. Bolina*) changes when the OSR is distorted due to a male-killing bacterial infection like *Wolbachia* and how this change in strategy affects the behaviour of females (their average mating rate) and the size of spermatophores males produce in successive matings.

In order to address this question models of both male and female behaviour were developed using elements of the models described in the considerable available literature [6,8,9,10,11]. The following sections discuss the modelling assumptions made and the method of dynamic programming [12] combined with simulation used to solve the model.

Model description

In the models the butterflies are treated as rational entities who are aiming to maximise their offspring. Consequently the language used in describing the model implies not only consciousness on the part of the butterflies but also the ability to devise and follow a strategy through mathematical analysis of the environment. This is clearly not the case in reality and the language of consciousness is used purely for convenience and to aid understanding of the model. It is assumed that under the pressure of natural selection the butterflies evolve to make instinctive behavioural decisions about mating which are optimal, and when viewed externally are equivalent to the mathematical operations described in the model.

The Female Model

Initially females are virgins with the potential to produce some maximum quantity of offspring. The number of offspring actually produced by each female is assumed to follow a 'diminishing returns' principle with respect to the amount of sperm she has gained. The proportion of the maximum offspring produced by a given female follows [8] and is given by the expression:

$$Offspring = \left(1 - e^{-\frac{sperm}{\alpha}}\right)$$

Where *sperm* is the level of sperm collected and α is a parameter which controls the steepness of the curve.

A female will quit the mating process if she has a certain amount of sperm. This parameter was set to 50% of the sperm carried by one male.

It is assumed that the sperm is allocated on a 'first come first served' basis such that the first male to contribute sperm benefits from the initial steep region of the curve, getting maximum return for the sperm investment, leaving the continually less well-performing regions of the curve for the late-comers. This automatically incorporates an unfair raffle principle analogous to that described by Parker [11] where late arriving sperm are effectively devalued relative to those already in place.

As time progresses, it is assumed that females who choose to continue seeking a mate will use some quantity of their resources to do so. It seems reasonable to suppose that this will lead to a decrease in the maximum offspring possible for that female. To account for this the above formula was modified to include a geometric 'depreciation factor' *d*:

$$Offspring = \left(1 - e^{-\frac{sperm}{\alpha}}\right) d^t$$

(where *t* is time.)

The females' strategy was static, presumed to have evolved over time through natural selection. The strategy is not necessarily optimal when considered against the various male strategies in play: a more realistic model would allow evolution of the female strategy towards some global optimum over time as the male and female strategies interact. However simulation of the co-optimisation of male and female strategies is beyond the scope of this essay.

The Male Model

The mating window for males and females is split into ten time steps. During each time step all butterflies mate at most once – i.e. males and females randomly pair up for mating in each time step; any 'spare' individuals caused by a non-even sex ratio

do not mate in that time step, and no butterfly mates more than once per time step. Under these conditions a strategy is defined as 'given a chance to mate at time step *t* and sperm reserve level *s*, allocate *q* sperm'.

At the beginning of the first time step males were assumed to have some fixed quantity of sperm, defined as 1.0, which they partition into some number of equal sperm quanta (100 sperm quanta for this essay[1]). During mating the male may choose to allocate any number of his remaining sperm quanta to the female, taking into account the current time step and sperm reserves. Fitness is measured by the number of offspring the male may expect from mating with females.

The dynamics of such a strategy is complicated. The expected payoff from a given mating depends on how much sperm the female in question has already collected: if mating with a virgin female then the male is benefiting from the steep region of the payoff curve and receives good return on his investment, while mating with a non-virgin female gives increasingly poor return on sperm investment.

In deciding how much sperm to 'spend' on a given female the male must also assess how likely he is to be able to mate in the future (based on some projection of the OSR into the future), and whether it is worth retaining some sperm for future mating opportunities when the payoff may have improved. For example since females choose to quit the mating pool once they have a certain threshold of sperm, they leave behind virgin (or at least less sperm-rich females) thus the sperm competition from the male point of view may decrease in successive time steps, making sperm conservation worthwhile. However this is a bad idea if it is highly unlikely for a male to have another opportunity to mate e.g. because it is the last time step being modelled, or because the OSR has shifted to make future mating unlikely.

Dynamic Programming

Optimisation of the male model lends itself conveniently to a process of dynamic programming described by Mangel and Clark [12]. This is an iterative algorithm where a decision of how much of some resource (sperm) to spend at a given time step depends on a dynamic state variable (sperm reserve) available to the decision maker at each time step. Dynamic programming algorithms characteristically work backwards from the last time step, optimising the resource allocation at each time step taking into account the knowledge of the future best-case outcomes calculated in the previous iterations of the algorithm.

For sperm modelling the problem arises in calculating the payoff at a given time step. As explained above this requires some knowledge of the expected quantity of sperm a female already possesses, and the probability of future mating opportunities based on the OSR. However the amount of sperm a female may be expected to have depends on the mating history of that female (i.e. what sperm decisions have been made in earlier time steps by other males). This information is unavailable to the algorithm as described since it is working backwards in time, not forwards.

[1] Varying this parameter would allow modelling of the degree of male control over sperm allocation per mating, which is clearly a biologically relevant parameter. Although not considered for the purposes of this essay, the impact of sperm quanta control might be an interesting course for further study.

In addition the OSR is known only at the beginning of the first time step. As females become satisfied with the sperm they have received, they quit the mating pool which lowers the OSR. Therefore OSR is not a static parameter in this model, but varies over time according to the behaviour of both males and females, and is therefore dependent on male strategy.

Simulation

In order to solve this problem an element of simulation is introduced in order to allow the algorithm to 'know' the expected quantity of sperm possessed by any female at any given time, and also the OSR at any time step.

The simulation assumes a fixed population of several thousand individuals, with an initial female:male OSR which is defined by the user. Males are given a strategy (as defined above) telling them how much sperm to use at a given mating.

For each time step:

1. Females with more sperm than the threshold level (0.5) leave the mating pool.
2. The OSR, based on the number of females remaining, is calculated and recorded.
3. The mean sperm level of each female is calculated and recorded.
4. Random pairs from the male and female populations are selected for mating until one of the populations runs out of individuals.
5. Within each mating pair, the male allocates sperm to the female according to the strategy.

The simulation provides the expected sperm competition and OSR at any time step, assuming the male population follows some given strategy. Running the dynamic programming algorithm will give the optimum strategy for one male to play against the population of males, assuming that the effect of the mutant's own strategy on the overall distribution of sperm per female and OSR is negligible.

By repeating the simulation with each male having the new strategy as defined by the dynamic program, a new time course of female sperm level and OSR is computed, which can be used by another iteration of the dynamic programming algorithm, to give a further optimised strategy. In this way the model should reach a point where the dynamic programming algorithm can no longer improve on the current male strategy, even with the prior knowledge of how everyone else behaves supplied by the simulation step.

The evolutionarily stable strategy (ESS) is that which, if adopted by a population, cannot be invaded by any competing alternative strategy [13]. Therefore if the dynamic programming and simulation process described above is allowed to iterate to convergence, the final strategy should be the ESS.

Loops in the optimisation process where the output oscillate between a number of strategies are perturbed to encourage convergence at a single solution corresponding

to the ESS; if the algorithm returns to these oscillating values after being perturbed it may be assumed that the oscillations represent multiple optimal strategies. In this case, the simulated population used to derive average sperm allocation and female mating rates is assumed to be composed of an even mix of the optimal strategies rather than a single ESS.

Summary of model parameters

Parameters which control the model are summarised in Table 1.

OSR	varied from 1.0 to 2.0 (steps of 0.1), 2.0 to 5.0 (steps of 0.5), 10 to 25 (steps of 5)
Alpha	0.5
Female satiety threshold	0.5
Female depreciation	10% and 0% reduction in maximum offspring per time step
Male depreciation	no depreciation
Sperm quanta	100
Time steps	10
Simulation population size	20,000

Table 1: Summary of model parameters

Implementation

The model was implemented in python. See Appendix for source code.

Results

An example strategy produced by the model is illustrated in Figure 1. The strategy indicates how much sperm to allocate given the current sperm level, and current time step.

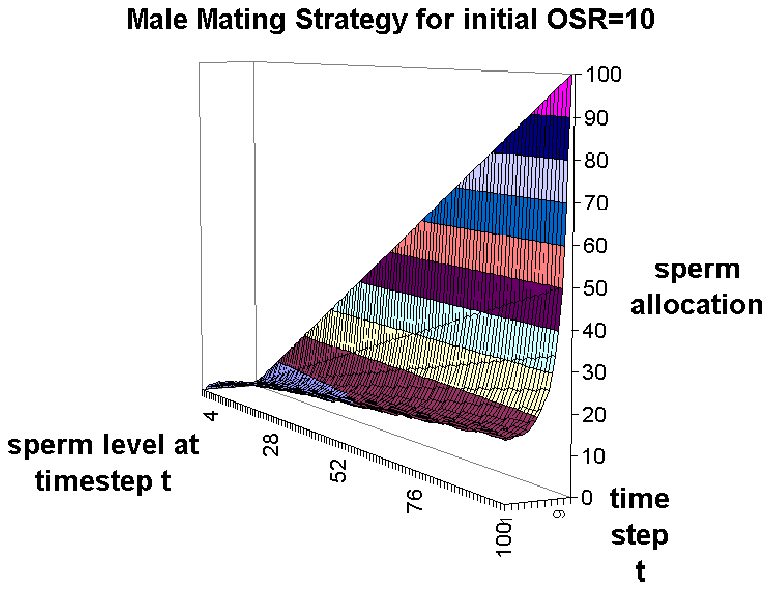

Figure 1: Example of an ESS produced by the model

Convergence to an ESS

At OSR between 1.0 and 2.0 the model tends to exhibit multiple solutions, while with the initial OSR set to above 2 females per male the algorithm rapidly finds an ESS.

OSR	number of solutions	notes
1.1	2	100% / 73% initial allocation
1.2	1	79% initial allocation (ESS)
1.3	2	78% / 70% initial allocation

Table 2: solution space for OSR close to 1.0

For OSR less than 1.3 (see Table 2) the solutions are clearly a trade-off between using all available sperm and gambling on being one of the few males who gets the chance to mate with the small surplus of females – in a population of males who always use all sperm at first opportunity, it is better to hold back and vice-versa. The population might arrive at some equilibrium of these two strategies, or more likely one will prove more favourable when some other factor is incorporated into the model.

For OSR above 1.3 and below 2.0 there start to appear a large numbers of optimal solutions, however on closer inspection they are almost all the same with very minor variations – e.g. a 1% difference in sperm allocation for the third time step when the male has 50 sperm quanta remaining. Since the model makes coarse approximations about butterfly behaviour, such minor variation in the output is unimportant. Although the algorithm fails to find an ESS in the strictest sense, in reality the multiple strategies would be indistinguishable from each other.

Average female mating rate

The average number of matings per female is illustrated in Figure 2.

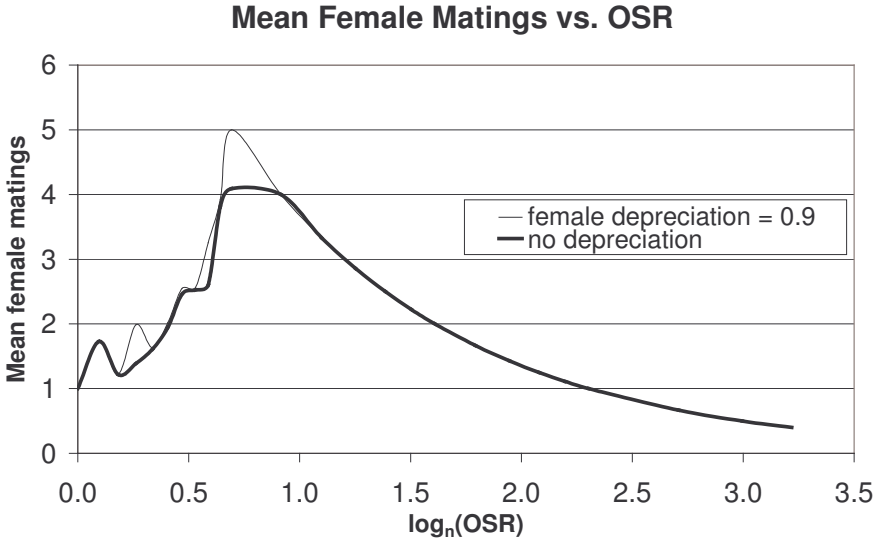

Figure 2: Mean copulations per female

This agrees with the results of [1] which show that as females become more abundant initially the mating rate rises, contrary to what one might expect, until males become so rare that sperm limitation takes place.

Spermatophore size

The average spermatophore size for successive mating events is shown in Figure 3 (model with female depreciation of 10% per time step) and Figure 4 (no female depreciation).

At low values of OSR males deliver most of their sperm at the first opportunity, while at high values of OSR they conserve some sperm for later mating events. For values of OSR higher than 10 the males are certain to find a mate in all time steps since it is impossible for the OSR to fall below 1 given the parameters of the model.

With female depreciation removed (i.e. less incentive to use sperm early due to drop in female quality) the ejaculate profile looks more even, as one would expect from intuition. In this case the reason for slightly higher sperm allocation early on is that sperm competition is zero at the first time step guaranteeing that the male is getting the benefit of the maximum slope of the payoff curve.

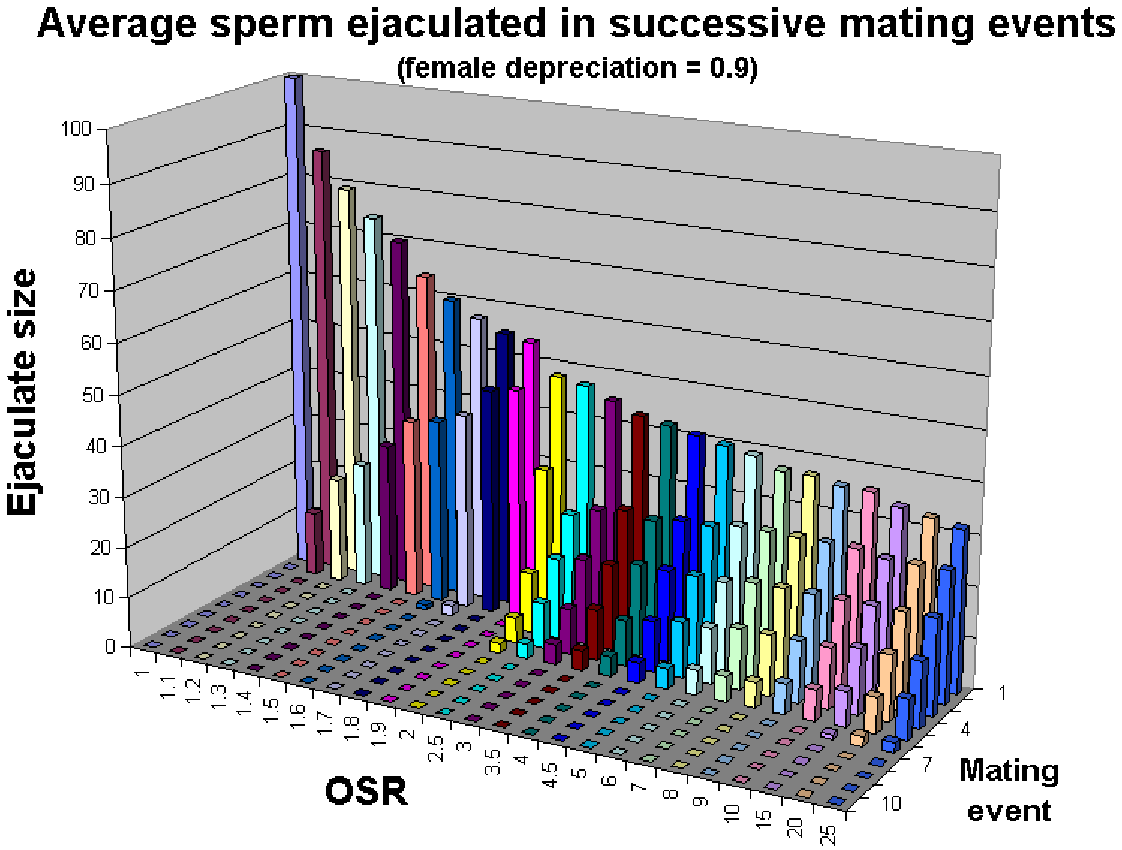

Figure 3

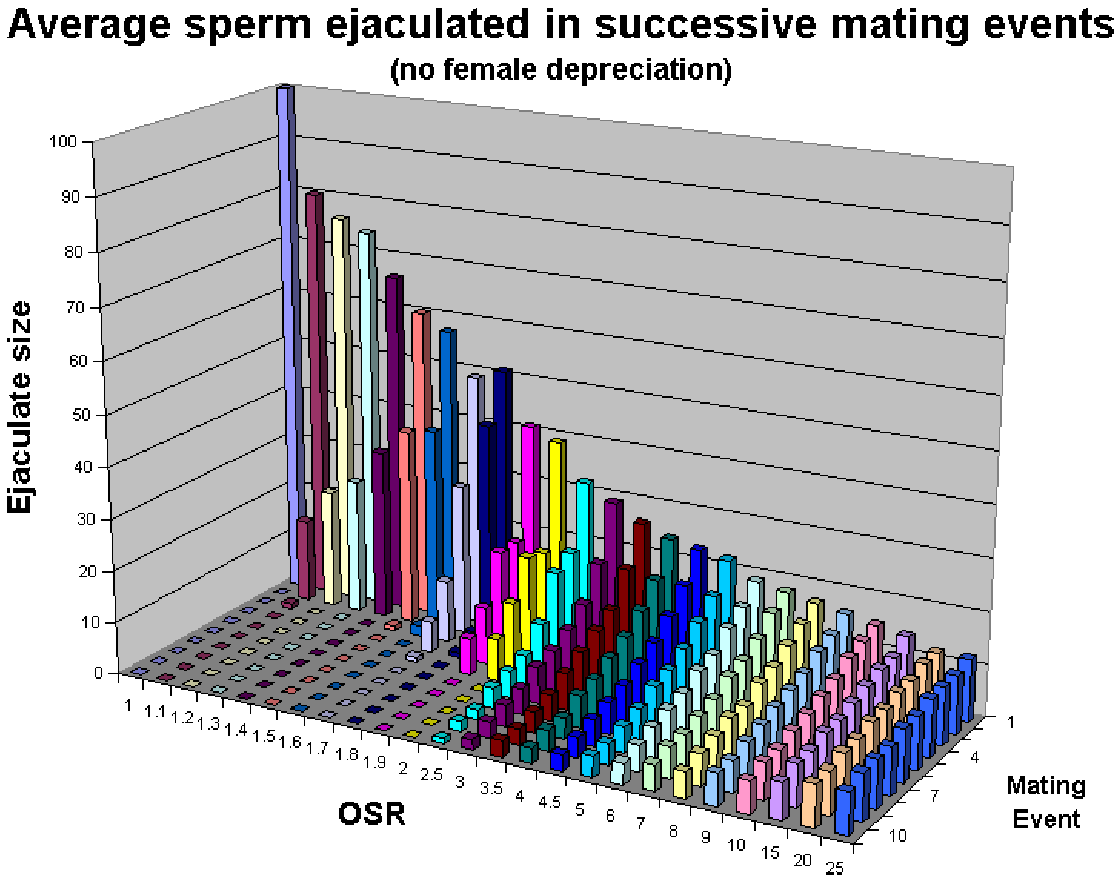

Figure 4

It is interesting to note that sperm allocation for the second mating increases as the OSR increases (corresponding to a second chance to mate becoming more likely). This causes an increase in sperm competition (see Figure 5) (measured as the fraction of all sperm that finds itself competing with another male's sperm for fertilization of a female's eggs).

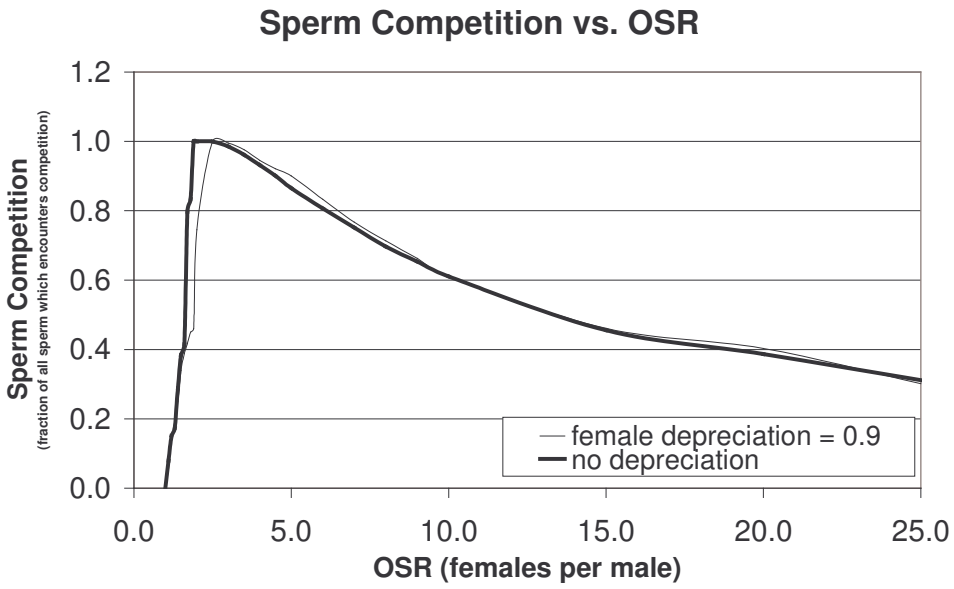

Figure 5: Sperm competition vs. operational sex ratio

This supports the suggestion of Williams et. al. [9] that sperm competition should be regarded as a consequence of sperm allocation strategy rather than as a driving force. Increased sperm competition, in this model, is the price males pay to benefit from the increase in mating opportunity provided by excess females.

Conclusions

The technique of dynamic simulation combined with simulation used in this essay allows models of butterfly mating behaviour to be solved, yielding optimal sperm strategies for a range of OSR values. Although the models presented are fairly simple with few parameters, the framework developed for this essay offers the possibility of investigating the complex interaction between conflicting female and male strategies. The results produced seem to correspond reasonably well with those found in the extensive literature on this subject.

Biologically speaking it is highly questionable whether one would expect real males in the wild to exhibit the optimised strategies calculated here. The distorted OSR produced by a bacterial parasite such as *Wolbachia* will persist for as long as it takes for the butterflies to develop some resistance to the parasite. It seems unlikely that enough mutations could occur to produce a phenotypic effect of the optimum sperm model in the time it takes for a mutation to confer parasitic immunity, even taking into account the amplification effect of male-inherited traits produced by the biased OSR.

Bearing in mind the above limitation the optimal strategies are useful to the biologist mainly as a guide to indicate the general direction in which 'sperm strategy evolution' would be expected to travel given enough time. Simulation of sperm strategies gives estimates of female mating rates and spermatophore size distributions which can be compared with experimental data, allowing the predictions of the model to be tested against measurable data.

References

1. Charlat, S., Reuter, M., Dyson, E.A., Hornett, E.A., Duplouy, A., Davies, N., Roderick, G.K., Wedell, N., Hurst, G.D.D. (2007). "Male Killing Bacteria Trigger a Cycle of Increasing Male Fatigue and Female Promiscuity." Current Biology *17*, 273-277
2. Parker, G.A. (1970) "Sperm competition and its evolutionary consequences in the insects." Biol. Rev. *45*: 525-567
3. Simmons, L.W., Siva-Jothy, M.T. (1998) "Sperm Competition in Insects: Mechanisms and the Potential for Selection", Sperm Competition and Sexual Selection, ed. Birkhead, T.R., and Møller, A.P., Academic Press, Ch10 pp 341-434
4. Cook, P.A., and Wedell, N. (1996) "Ejaculate dynamics in butterflies: a strategy for maximizing fertilization success?" Proc. R. Soc. Lond. B *263*, 1047-1051
5. Wedell, N., Matthew J.G.G., Parker, G.A. (2002) "Sperm competition, male prudence and sperm-limited females" TRENDS in Ecology & Evolution *17* 7 313-320
6. Harris, W.E., and Lucas, J.R. (2002) "A state-based model of sperm allocation in a group breeding salamander" Behavioral Ecology *13* 5 705-712

7. Ball, M.A., and Parker, G.A. (2007) "Sperm competition games: the risk model can generate higher sperm allocation to virgin females" Journal Compilation European Society for Evolutionary Biology *20* 767-779
8. Galvani, A., Johnstone, R. (1998) "Sperm allocation in an uncertain world" Behav. Ecol. Sociobiol. *44* 161-168
9. Williams, P.D., Day, T., Cameron, E. (2005) "The evolution of sperm-allocation strategies and the degree of sperm competition" Evolution *59* 3 492-499
10. Fryer, T., Cannings, C., Vickers, G.T. (1999) "Sperm Competition I: Basic Model, ESS and Dynamics" J. theor. Biol. *196* 81-100
11. Parker, G.A. (1990) "Sperm competition games: raffles and roles" Proc. R. Soc. Lond. B *242* 120-126
12. Mangel, M., and Clark, C.W. (1988) "Dynamic Modeling in Behavioral Ecology" Princeton University Press
13. Maynard-Smith, J. (1982) "Evolution and the theory of games" Cambridge University Press

Appendix (model source code)

```
# butterflies.py
# dynamic programming and simulation of butterfly
# sperm allocation strategy.
# David Fallaize CoMPLEX, UCL 2007

import sys
import math, random
import copy
from simulator import *
from outputs import *

def chop(x, lower, upper):
    if x > upper: x = upper
    if x < lower: x = lower
    return x

def frange(start, end=None, inc=None):
    "A range function, that does accept float increments..."

    if end == None:
        end = start + 0.0
        start = 0.0

    if inc == None:
        inc = 1.0

    L = []
    while 1:
        next = start + len(L) * inc
        if inc > 0 and next >= end:
            break
        elif inc < 0 and next <= end:
            break
        L.append(next)

    return L

def find_best_strategy(simulation, T):
    # this is the dynamic programming algorithm

    new_strategy = {}
    t = T
    while t > 0:

        new_strategy[t] = {}

        # for each of the possible sperm levels, calculate the best
        # sperm allocation choice based on expected payoff.
        sperm_competition = simulation.expected_female_sperm_count(t-1)
        current_payoff = Female().calc_payoff(sperm_competition, t)
        #threshold_allocation = Female.threshold(t) - sperm_competition

        for s in range(Male.sperm_quanta+1): # s from 0 to 1

            # given that i currently have s sperm available, iterate through
            # my possible amounts to allocate, and decide whether it's better
            # to allocate that sperm now, or later - i.e. find the best total
            # payoff of sperm now + payoff that leaves us for next timestep.
            def payoff(a):

                # if i allocate amount of sperm a, that will leave me with
                # s-a for the next time step, minus depreciation due to cost
                # of flying around looking for mate. We have already worked out
                # what the best payoff for that is.

                # best future payoff is the best we can do with this amount of
                # sperm in the next step, taking into account the odds of actually
                # being able to mate in the next step by using the simulated OSR
                sperm_next_time = math.floor((s-a) * Male.depreciation)
                future = 0.0
                try:
                    accumulator = 1.0
                    for tt in range(t,T):
```

```
                        prob_of_mating = chop(simulation.OSR[tt+1],0,1)
                        future += new_strategy[tt+1][sperm_next_time][1] *
prob_of_mating * accumulator
                        accumulator *= (1.0 - prob_of_mating)
                except:
                    if tt < T: print "argh"
                    pass

                now = Female().calc_payoff(sperm_competition + a, t) - current_payoff
                return now + future

            best_now = max([(payoff(a),a) for a in range(s+1)])
            new_strategy[t][s] = (best_now[1], best_now[0])

        # decrement t
        t -= 1

    # strategy as-is includes payoff information which is now
    # redundant...
    new_strategy_stripped = {}
    for t in new_strategy.keys():
        q = {}
        for s in new_strategy[t].keys():
            q[s] = new_strategy[t][s][0]
        new_strategy_stripped[t] = q

    return new_strategy_stripped

# here's where we actually run the program

# parameters!
T = 10
sample_size = 5000
osr_range = frange(1,2,0.1) + frange(2,5,0.5) + range(5,10,1) + range(10,26,5)
#osr_range = [1.1, 1.2, 1.3, 1.6] # non-converging OSR values

# init holders
ejaculate_sizes = {}
spermatophore_sizes = {}
sperm_competition = {}
female_mating = {}

def random_strategy():
    # generate a random strategy

    strat = {}
    for t in range(1,T+1):
        strat[t] = {}
        for s in range(1,Male.sperm_quanta+1):
            strat[t][s] = int(math.ceil(random.random() * s))

    return strat

def perturb_strategies(strats):
    # get 'average' strategy of set of looping strategies

    new_strategy = {}
    for t in range(1,T+1):
        new_strategy[t] = {}
        for s in range(1,Male.sperm_quanta+1):
            new_strategy[t][s] = 0
            for strat in strats:
                new_strategy[t][s] += strat[t][s] / len(strats)

    return new_strategy

# loop over range of initial OSR
for OSR in osr_range:
    print "OSR=%f" %(OSR)

    new_strategy = random_strategy()
    tried_strategies = [new_strategy]
    perturbed_list = []
    converged = False
    iterations = 0
    while not converged:
        iterations += 1
```

```
        # statistical simulation
        simulation = Population(sample_size, OSR, T, new_strategy)

        # generate new strategy by dynamic programing
        new_strategy = find_best_strategy(simulation, T)

        # check for convergence
        print "testing convergence...",
        if new_strategy not in tried_strategies:#

            # count how many discrepencies
            discrepencies = 0
            for t in range(1,T+1):
                for s in range(1,Male.sperm_quanta+1):
                    if new_strategy[t][s] != tried_strategies[-1][t][s]:
                        discrepencies += 1
            print "failed: %d discrepencies" %(discrepencies)

            # add tried strategy to the list
            tried_strategies.append(copy.deepcopy(new_strategy))

        elif new_strategy == tried_strategies[-1]:
            # converged! same strategy emerges as the last one
            print "succeeeded!"

            # output all the interesting population statistics stuff
            write_strategy(new_strategy, OSR)

            ejaculate_sizes[OSR] = [simulation.ejaculate_sizes()]
            spermatophore_sizes[OSR] = [simulation.spermatophore_sizes()]
            sperm_competition[OSR] = [simulation.sperm_competition()]
            female_mating[OSR] = [simulation.average_female_matings()]

            converged = True

        elif new_strategy not in perturbed_list:
            # seen this strategy already, but haven't tried perturbing

            print "already seen this strategy... try perturbing"
            idx = tried_strategies.index(new_strategy)
            strats = tried_strategies[idx:]
            new_strategy = perturb_strategies(strats)
            perturbed_list += strats

            tried_strategies = tried_strategies[:idx]
            tried_strategies.append(copy.deepcopy(new_strategy))

        else:
            # seen this strategy, tried perturbing it, still end up looping
            # therefore must be multiple stable states

            idx = tried_strategies.index(new_strategy)
            print "loop! strategies: %d-%d are optimal" %(idx+1,
len(tried_strategies))
            ctr = 0

            for a_solution in tried_strategies[idx:]:
                ctr+=1
                write_strategy(a_solution, OSR, ctr)

            simulation = Population(sample_size, OSR, T, tried_strategies[idx:])
            ejaculate_sizes[OSR] = [simulation.ejaculate_sizes()]
            spermatophore_sizes[OSR] = [simulation.spermatophore_sizes()]
            sperm_competition[OSR] = [simulation.sperm_competition()]
            female_mating[OSR] = [simulation.average_female_matings()]

            converged = True

    print "converged after %d iterations" %(iterations)
    write_csv("spermatophore_size", spermatophore_sizes)
    write_csv("ejaculates", ejaculate_sizes)
    write_csv2("sperm_competition", sperm_competition)
    write_csv2("female_mating", female_mating)

# simulator.py
# dynamic programming and simulation of butterfly
```

```
# sperm allocation strategy.
# David Fallaize CoMPLEX, UCL 2007

import sys
import math, random

class Population:

    def __init__(self, size, OSR, T, strategy):

        self.T = T
        self.strategy = strategy

        # OSR is the number of females per male
        males = int(math.floor(size / (1 + OSR)))
        females = int(size - males)
        F = [Female() for i in range(females)]

        # handle multiple strategies in population
        if type(strategy) is list:
            self.M = []
            males_per_strategy = math.floor(males / len(strategy))
            for s in strategy:
                self.M += [Male(s) for i in range(males_per_strategy)]
        else:
            self.M = [Male(strategy) for i in range(males)]

        self.OSR = {}
        self.sperm_histogram = {}
        self.final_females = []

        # given this ratio of males to females, and the optimum male
        # strategy as supplied, calculate the OSR vs t
        for t in range(T+1):

            # construct the list of active females. Note that males never
            # leave the mating pool as it is always in their interest to run
            # interference against other males who want to mate in order to
            # reduce any chance of sperm competition.
            self.final_females.extend([f.sperm_holder for f in F if not
f.still_mating(t)])
            F = [f for f in F if f.still_mating(t)]

            females = len(F)
            males = len(self.M)
            self.OSR[t] = float(females) / males

            # construct sperm histogram for this time point
            self.sperm_histogram[t] = [f.sperm_count for f in F]

            # now carry out the mating for this time step
            if t < T:
                n = min(males, females)
                for f,m in zip(random.sample(F, n), random.sample(self.M, n)):
                    f.add_sperm(m.give_sperm(t+1))
                for m in self.M: m.depreciate() # allow male depreciation

            # that completes one time step (i.e. one iteration of mating)
            # at the start of the next time step (t+1) the females have the
            # opportunity to quit if they think they have enough sperm.

        self.final_females.extend([f.sperm_holder for f in F])

    def expected_female_sperm_count(self, t):
        try:    expect = float(sum(self.sperm_histogram[t])) /
len(self.sperm_histogram[t])
        except: expect = 0
        return expect

    def average_female_matings(self):
        return float(sum([len(spermatophores) for spermatophores in
self.final_females])) / len(self.final_females)

    def sperm_competition(self):
        total_sperm = sum([sum(spermatophores) for spermatophores in
self.final_females])
```

```
        in_competition = sum([sum(spermatophores) for spermatophores in
self.final_females if len(spermatophores) > 1])
        return float(in_competition) / total_sperm

    def ejaculate_sizes(self):

        # init counters
        hist,ctr = {},{}
        for t in range(1,self.T+1):
            hist[t] = 0.0
            ctr[t] = 0.0

        # for each of the males...
        for m in self.M:
            # loop over their mating events, and build average
            # sperm allocation for sequential mating events
            for idx in range(len(m.mating_history)):
                hist[idx+1] += m.mating_history[idx]
                ctr[idx+1] += 1

        # calculate averages
        output = []
        for t in range(1,self.T+1):
            if ctr[t] != 0:
                output.append(hist[t] / ctr[t])

        return output

    def spermatophore_sizes(self):

        # init counters
        hist,ctr = {},{}
        for t in range(1,self.T+1):
            hist[t] = 0.0
            ctr[t] = 0.0

        # for each of the females...
        for f in self.final_females:
            # loop over their mating events, and build average
            # sperm allocation for sequential mating events
            for idx in range(len(f)):
                hist[idx+1] += f[idx]
                ctr[idx+1] += 1

        # calculate averages
        output = []
        for t in range(1,self.T+1):
            if ctr[t] != 0:
                output.append(hist[t] / ctr[t])

        return output

class Male:

    total_sperm = 1.0
    sperm_quanta = 100
    depreciation = 1.0 # no male depreciation taken into account

    def __init__(self, strategy):
        self.sperm_count = Male.sperm_quanta
        self.strategy = strategy
        self.mating_history = []

    def quanta_to_sperm(cls, q):
        return (q / float(cls.sperm_quanta)) * cls.total_sperm
    quanta_to_sperm = classmethod(quanta_to_sperm)

    def give_sperm(self, t):

        # figure out how much sperm we should allocate according
        # to the current strategy we're following.  Note that this
        # will depend on both the time t and the amount of sperm we
        # currently have available (the totally ideal strategy might
        # not be possible if females are in short supply, and males
        # do not have the opportunity to mate every time they would like)
        if self.sperm_count > 0:
            sperm_to_donate = self.strategy[t][self.sperm_count]
```

```
        else:
            sperm_to_donate = 0

        self.sperm_count -= sperm_to_donate
        self.mating_history.append( sperm_to_donate )

        return sperm_to_donate

    def depreciate(self):

        # cause a depreciation in sperm due to 'cost of living'
        self.sperm_count = math.floor(self.sperm_count * Male.depreciation)
        return

class Female:

    sperm_potency = 0.5
    depreciation = 0.9
    threshold = 0.5

    def __init__(self):
        self.sperm_holder = []

    def add_sperm(self, sperm):
        self.sperm_holder.append(sperm)

    def sperm_count(self):
        return sum(self.sperm_holder)
    sperm_count = property(sperm_count)

    def calc_payoff(sperm, t):

        # convert sperm quanta to an actual amount of sperm
        actual_sperm = Male.quanta_to_sperm(sperm)

        # depreciation of female quality over time
        dep_factor = Female.depreciation ** (t-1)

        return (1 - math.exp(-actual_sperm / Female.sperm_potency)) * dep_factor
    calc_payoff = staticmethod(calc_payoff)

    def calc_still_mating(sperm, t):
        return Male.quanta_to_sperm(sperm) < Female.threshold
    calc_still_mating = staticmethod(calc_still_mating)

    def still_mating(self, t):
        return self.calc_still_mating(self.sperm_count, t)
```