Population genetic inference of demographic processes in the African Wild Silk Moth, *Gonometa postica* (Lasiocampidae)

by

Wayne Delport

Supervisors
Prof. Paulette Bloomer
Prof. J. Willem H. Ferguson

Submitted in fulfillment of the requirements for the degree Philosophiae Doctor in the Faculty of Natural and Agricultural Science, University of Pretoria, Pretoria

June 2005

Declaration

I declare that this dissertation, which I hereby submit for the degree Philosophiae Doctor at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

13 December 2005

For Linda

Gonometa postica

female

male

Summary

The African Wild Silk moths (*Gonometa* spp., Lasiocampidae) are species that are presently of particular economic interest in southern Africa. Both *Gonometa postica* and *G. rufobrunnea*, two species of African Wild Silk moth native to southern Africa, have been shown to possess a silk fibre of exceptional quality. A small-scale cottage industry utilizing the silk of *Gonometa* species currently exists in southern Africa, yet a consistent complaint is the lack of supply of cocoons. The *Gonometa* species in southern Africa have been shown to exhibit large inter-annual population fluctuations. However, it is uncertain whether eruptions are only the result of local populations experiencing ideal conditions or whether current eruptions are initiated by dispersal of individuals from eruptive populations in previous generations. A second observation, regarding eruptions, is that they are patchily distributed at both the local (within outbreaks) and regional scale (across southern Africa).

In this thesis I have studied population eruptions through distribution analysis of three years of presence/absence data, and through spatial and temporal population genetic analysis. The analysis of population genetic data allows the inference of population demographic parameters such as population size fluctuations and migrations. In particular, the use of microsatellite markers allows a high-resolution analysis of the connectivity of populations, and provides signal of population size fluctuations. I utilise both mitochondrial DNA control region sequences and polymorphic microsatellite loci to make inferences of population processes in *G. postica,* using a combination of both analytical and simulation model analysis approaches. The results, in general, indicate that dispersal of moths across South Africa is extensive. These results are further considered in light of the effects of population size fluctuations on spatial genetic pattern, where the potential exists for unstable population demography to influence the inference of dispersal from population genetic data. The population genetic analyses presented here allow the inference of the extent of a local population/outbreak, and the degree of movement between local populations. Given that a large-scale population dynamics project based on *G. postica* is currently under development, the results determine the geographical extent at which the population dynamics study should be conducted. Furthermore, the population genetics data

generated will contribute to the construction of a population dynamics model, including abiotic and biotic variables, which will allow a better understanding of eruptions in this species.

Acknowledgements

"But thanks for your time
Then you can thank me for mine
And after that's said
Forget it"

Rodriguez

Perhaps a little more blunt than I intended, but so many need to be acknowledged that I have approached this section with much restraint. I owe so much gratitude to my advisors, Paulette Bloomer and Willem Ferguson. Paulette and Willem have quietly instilled in me an obsession with population genetics. I think they saw the sparkle in my eye at the mere mention of working below the level of the species, and have done everything necessary to develop this mild interest into a maddening obsession. Paulette has created an academic environment that encourages creativity, and for this I am extremely grateful. Willem put me behind a computer, bought me a Delphi software license and let me be. Thanks Willem, I've learnt so much. The interaction with international post-doctoral fellows, Michael Cunningham and Kelley Whitaker, has been extremely fruitful and has created a dynamic academic environment for the research group. Michael and Kelley have demonstrated that there are Australians worth talking to; "no worries mate!" I owe Michael Cunningham more thanks than can be expressed with mere words. I have learnt so much since we first started talking about coalescence and population genetics. Thanks Michael. The rest of MEEP (Molecular Ecology and Evolution Programme) have been great. Thanks to listening to my ramblings in journal club. To Isa-Rita Russo, Carel Oosthuizen, Arrie Klopper, Sarita Maree, Catherine Sole, Maria-Noel Cortinas, Joha Grobbelaar, Lucille Hermann, Ernst Swartz and Jonathan van Alphen-Stahl I have the following to say: The only way to familiarize oneself with complex ideas is to discuss them. We've had great discussions in journal club. Keep it up and we'll all reap the rewards. I owe special thanks to Isa-Rita Russo (aka bench-babe) for keeping me warm on that cold winter night in Windhoek whilst sampling moths. Not quite what it sounds like, but Isa knows the punch line. Isa you've been a great friend in the lab and office. Thanks. I'm sure one day we'll happily reflect on our graduate student days from the comfort

of our permanent university posts (I hope). Ernst Swartz, I guess I should also thank you for keeping me warm on that cold night in Venda. Definitely not at all what it sounds like. Ernst, we've had a few good grogs (beers) and great discussions. Thanks. Tony Knowles, I've missed the good grogs we shared before you moved to Stellenbosch. I know we will still publish a manuscript from the Brincadian Institute of Science. Thanks for great discussions in our undergraduate and early graduate years. They have culminated in where I stand today, thesis in hand and a mind packed full of ideas. Thanks boet.

I have had such great opportunities in the last three years and again I have my advisors, Paulette Bloomer and Willem Ferguson, to thank. Through the support of the Mellon Foundation Programme, at the University of Pretoria, I was able to attend an international conference (European Society of Evolutionary Biology 2003), a workshop on molecular evolution (http://workshop.molecularevolution.org/) at Woods Hole, MA, and gain experience in an international silk moth genetics laboratory (Dr J. Nagaraju, Centre for DNA Fingerprinting and Diagnostics, Hyderabad, India). Without this funding these experiences would not have been possible. Thanks to Willem and Paulette for encouraging my development as an academic, and for utilizing the Mellon funds exclusively for this purpose.

Moving away from the academic realm I have my close friends, Darryl and Tracy Adriaanzen, Andrew and Gina Smith, Ralph & René (soon to be) Tudhope, to thank. You guys have been so supportive and have always had absolute confidence in my abilities to pursue my academic endeavors. Yes we will enjoy that bottle of Johnny Walker Blue, let me first find a job though. Maybe you'll see me on John Vorster drive with my sign: "PhD Genetics, no work, no soap, no kids to feed. Please help!" From friends to family, Mom, Dad, and boet Sean: you have supported me and have always believed that I could accomplish what I initially set out to do. Thank you. The love and support from you has been unconditional, as it has been from our Father. Bradley, my nephew, I will kick your butt at Need for Speed. Maybe now I have time to practice. To my other parents, Kaki and Rina, thanks for believing in me and for entrusting your daughter to an unemployed scientist. I'm not sure I would've done likewise. My sisters, Jax and Tashi, you guys are great. Somehow you thought it was cool that I was still at university. Thanks. Jax, I worry about you. I see that academic

glow in you. Don’t do it. Leave now, get a job, a big screen tv, and two weeks annual leave. No do it, and give it all you have. I know you have the potential. Finally, to my wife, Linda: you have supported me, loved me and embraced me. Thanks for putting up with late nights in the office, late nights in the lab and late nights at home. Linds, your support has been exceptional. Also thanks for listening to my ramblings, thanks for believing that my ramblings made sense, thanks for encouraging me to get my ramblings down on paper, and thanks for reading my ramblings. Now I can submit my ramblings and continue with our life together.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

"By morning the wind had brought the locusts; they invaded all Egypt and settled down in every area of the country in great numbers. Never before had there been such a plague of locusts, nor will there ever be again. They covered all the ground until it was black. They devoured all that was left after the hail—everything growing in the fields and the fruit on the trees. Nothing green remained on tree or plant in all the land of Egypt."

Exodus 10: 13-14

Population genetics and phylogeography have been shown to be indispensable in the understanding of species demographics (Avise *et al.* 1987, Knowles & Maddison 2002, Knowles 2004). The purpose of this dissertation is to apply population genetics principles to gain a better understanding of demographic processes in the African Wild Silk Moth (*Gonometa postica*). Therefore, the purpose of this introductory chapter is firstly, to review the known biology of the African Wild Silk Moth and secondly, to introduce the necessary population genetic principles and methods that will be used for demographic inference in later chapters.

The African Wild Silk Moth is a species that is currently of economic interest in southern Africa. Both this species and its sister species, *G. rufobrunnea,* have been shown to possess a silk fibre of exceptional quality (Freddi *et al.* 1993, Akai *et al.* 1997). In this respect the initiation of an African Wild Silk Industry in southern Africa has been proposed as a potential means of poverty alleviation in rural southern Africa. However, a consistent complaint from small-scale cottage industries that currently utilize *Gonometa* silk is the insufficient supply of cocoons. Since the industry currently only utilizes cocoons from which the adult moths have already emerged, there is little or no effect of harvesting on the population dynamics of the species. Rather, the insufficient supply of cocoons is directly related to the complex population cycles experienced by the species. The species is characterized by two generations per year, the first starting in September-October when adult moths emerge from cocoons. Adult moths emerge without feeding mouthparts and survive for three to five days (maximum nine days, Hartland-Rowe 1992) during which breeding occurs. Eggs are laid, larvae emerge and pass through six instar larval stages in approximately five weeks, after which the larvae construct cocoons, pupate and enter a period of diapause. This period of diapause either carries through to the following September when adults emerge, or is broken in February with adult emergence and an additional population cycle. Typically, this second generation comprises between 12-50% of the first generation (Hartland-Rowe 1992), and culminates in pupae that emerge as adults in September. *G. postica* experiences large inter-annual population size fluctuations (Veldtman 2004), though it is uncertain which factors contribute to the cyclical nature of this species.

In order to understand eruptions of this species several questions need to be addressed. Firstly, the influence of climatic factors on the incidence of population eruptions should be evaluated. It has been hypothesized that *G. postica* eruptions follow periods of drought, where the rates of larval parasitism are reduced during these times, thus allowing the normally heavily-parasitised larvae (Veldtman 2004) to reach eruptive proportions (Hartland-Rowe 1992). Secondly, the interaction between host plant, *Acacia erioloba,* phenology and *G. postica* is unknown. *A. erioloba* experiences a leaf-flush in August prior to the emergence of adult moths (Smit 1999). The timing of this leaf-flush in relation to rainfall, and the timing of *G. postica* emergence, is crucial for the understanding of eruptions in this species. Related to the phenology of the host plant, it is necessary to determine the quantity and quality of foliage required for larvae to complete development, pupation and emergence as adult moths. This interaction between climate, host-plant and *G. postica* would be crucial for the understanding of complex population dynamics in this species. Ideally, a long-term population dynamics programme should be initiated that evaluates the effect of both exogenous and endogenous factors in determining local annual population sizes of this species. The scale, however, at which such a study should be conducted is uncertain since there are currently no dispersal estimates available, and therefore it is uncertain what constitutes a population in this species. Such knowledge of dispersal ability will further enhance the interpretation of the temporal occurrence of eruptions, where eruptions in later years are sourced from nearby eruptions in preceding years. The purpose of this dissertation is to estimate the degree of genetic connectivity between eruptions within and between years, using spatial population genetic analysis. Such an understanding, of dispersal ability in this species, will allow the planning of harvesting strategies and potentially allow the incorporation of a dispersal parameter into predictive distribution modeling that is planned for future research.

Several species exhibit complex population cycles and large fluctuations in both density and population size (Finerty 1979, Bjornstad *et al.* 2002, Tallmon *et al.* 2002, Turchin 2003). Turchin (2003) has reviewed the dynamics of such complex population cycles and notes that such cycles are the result of the interaction between endogenous and exogenous factors. These factors encompass environmental effects, including climatic factors, population-specific effects such as density dependence and inter-species interactions, such as predator-prey relations. Although the ecological

literature of complex population cycles is extensive, comparatively few genetic studies have been conducted. Population genetic analysis of snowshoe hare (*Lepus americanus*) have attributed the observed spatial patterns of genetic variation to a stepping stone model of gene flow influenced by density cycles, where local bottleneck populations expand to previously unsuitable habitat and thus homogenize genetic diversity across the distribution (Burton *et al.* 2002). The collared lemming, another cyclic species, also has a spatial genetic pattern characterized by very little population structuring. In this species, the inferred high levels of gene flow are attributed to long-distance dispersal events (Ehrich *et al.* 2002). Similarly, deviation from an isolation by distance model in spatial genetic structure in the butterfly *Aglais urticae* is attributed to high movement rates, occasional long-distance migration and rare extinction/recolonisation events (Vandewoestine *et al.* 1999). Northeastern Australian rabbit populations also show large degrees of population size fluctuations, yet differ in degree of stochasticity between the arid west and semiarid east (Fuller *et al.* 1997). Spatial genetic patterns in this species corroborate the observation of a high degree of gene flow inferred from population genetic data of cyclical species in that the western populations exhibit reduced levels of structure versus that of the eastern populations, which have fewer stochastic fluctuations in population size (Fuller *et al.* 1997). This general result of very little population structuring in cyclical species may be worthy of further investigation (Burton *et al.* 2002). Theoretically, one might expect a higher degree of population sub-structuring in species that exhibit population cycles (Wright 1940), due to increased probability for different demes to become fixed for alternate alleles, under random genetic drift during periods of small population size. However, this effect is most likely dependent on the levels of dispersal and whether fixation of alleles at particular demes can be removed due to movement of alleles between demes in years of population size expansion. The effects of extinction and recolonisation on spatial genetic pattern have been evaluated for species with metapopulation structure (Wade & McCauley 1988, Whitlock & McCauley 1990, Ibrahim *et al.* 2000, Ibrahim 2001). These results in general indicated that the effect of population turnover on genetic differentiation is dependent on the number of individuals colonizing a deme relative to the number of recurrent migrants between demes (Whitlock & McCauley 1990, Ibrahim 2001). This is intuitive for metapopulations since low numbers of founders are likely to produce greater genetic structure, as is colonization from single versus multiple demes.

Recurrent migrations will furthermore tend to homogenize genetic diversity given high levels of migration. These results generally appear to hold for metapopulations, yet some species do not have obvious metapopulation structure and simply exist as continuous populations where neighbourhood sizes fluctuate as a result of local changes in density. Dispersal in this instance does not occur between spatially defined demes, but are rather effected as an individual dispersal distance in a continuously distributed isolation by distance model. The effects of population size variations in such species are likely to be different, and thus are explored within the context of this dissertation.

Inference of population demography from genetic data

In order to address the dispersal ability of *G. postica* through spatial genetic analysis it is necessary to use the currently available analysis approaches for spatial genetic data. The purpose of the following section is to introduce the available methods. This review of spatial genetic analysis methods is by no means exhaustive. Rather it is a personal reflection on the development of the field and is biased towards simulation and coalescent modeling approaches. Furthermore, although I have utilized spatial autocorrelation analyses methods in the subsequent chapters I have not covered these here. I feel that spatial autocorrelation approaches do not contribute to the development of custom demographic analysis models that I personally believe is the future of spatial population genetic analysis.

The inference of demographic parameters from population genetic data is a field that has grown rapidly in past years. This field, originally termed phylogeography (Avise *et al.* 1987) originated in the 1980's and has enjoyed a long tradition of gathering spatial genetic data, and subsequently inferring processes from the correlation of such data with landscape, or historical geographical/climatic features and events (see reviews by Avise *et al.* 1987, Avise 2000). This spatial pattern matching, however, is fraught with ideological and theoretical problems, the most notorious being the inference of complex demographic processes from a single gene tree. As such phylogeography and analyses of spatial genetic data has moved from a pattern-based descriptive science to one that involves the statistical testing of alternate hypotheses against the observed genetic data. The rapid increase in computer power in the last

decade has fueled this development of statistical phylogeography, which has been the subject of a recent special issue of *Molecular Ecology* (Volume 13, 2004). In this introduction I briefly review the history of spatial population genetic analysis and introduce the current advances in statistical phylogeography, and demographic inference. The purpose of this discussion is to provide a framework for the analysis of spatial population genetic data collected from the focal species of this dissertation.

Summary statistic approaches

Probably the most important consideration for any analysis technique is the particular demographic/migration model on which a particular technique is based. Three models are common in the population genetics literature, the island model (Wright 1940, Crowe 1986), the stepping-stone model (Kimura & Weiss 1964, Nagylaki 1982), and the isolation by distance model (Wright 1943), where a suite of summary statistics characterizes each. Historically, population genetics, and thus inference of demographic parameters such as migration, has been strongly based on summary statistics. Wright (1951) devised an approach to partitioning genetic variation in a subdivided population based on the island model of migration, termed Fixation indices or *F*-statistics. The calculation of F_{ST} has dominated the population genetics literature, and is simply the variance in allele frequencies across populations (V_a) standardized by the mean allele frequency, ρ.

$$F_{ST} = \frac{V_a}{\rho(1-\rho)} \qquad \text{Wright (1951)}$$

Several methods are available to estimate F_{ST} (Wright 1951, Weir & Cockerham 1984, Nei 1987), yet most make use of the relationship above. However, Slatkin (1985, 1987) has suggested an approach based on the distribution and frequency of rare alleles, and Barton and Slatkin (1986) have found these alternate measures to be consistent over a wide range of assumptions of population structure, selection and mutation. Recent reviews of F_{ST}, as a measure of population differentiation, should be consulted for further discussion (Weir & Hill 2002, Excoffier 2003).

Of particular importance in calculating F_{ST} are the underlying assumptions of the model, i.e. an island migration model in a Wright-Fisher population of constant size.

The principal aim of calculating F_{ST} is the inference of both effective population size (N_e), and migration rates between demes (m);

$$N_e = \frac{Nd}{1 - F_{ST}} \qquad \text{Wright (1931)}$$

$$F_{ST} = \frac{1}{4Nm + 1} \qquad \text{Wright (1931)}$$

where N = population size, d = the number of demes and m = migration rate. However, considerations of the assumptions, in terms of underlying population structure, of such calculations are paramount. Indeed, Whitlock (2004) has considered the application of the above estimators of N_e and $4Nm$ applied to metapopulations. A critical assumption in the above model is that of no variance in reproductive success among demes. Since F_{ST} typically takes on values between 0 and 1, the above estimator of N_e gives the nonsensical result of N_e always being greater than Nd, the product of population size per deme, N, and number of demes, d (Whitlock 2004). This result is contrary to what is expected is natural metapopulations, where large variances in reproductive success among demes is expected, and thus highlights the importance of considering the assumptions behind a particular model when analyzing and interpreting data. Several theoretical population models have been developed for the estimation of F-statistics and the analysis of population structure, including extinction-recolonisation metapopulation models (Whitlock & McCauley 1990), source-sink models (Gaggiotti 1996), and stepping-stone models (Kimura & Weiss 1964). However, since the purpose of many population genetic studies is to infer the underlying genetic structure of the focal species, the suitable model is not known *a priori*. Thus the development of statistical procedures that can simultaneously estimate the underlying genetic structure of a population, and demographic parameters, is a central challenge in population genetics (Excoffier 2003).

Given a continuous population the isolation by distance model (Wright 1943) is most appropriate, where summary statistics of interest are those concerned with neighbourhood size. Neighbourhood size essentially represents the number of individuals an individual would encounter within its lifetime, and is dependent on density (D) and the standard deviation of the distribution of dispersal distances (σ).

The size of a neighbourhood (N_b) amounts to the number of individuals in a circle with a radius twice the standard deviation of dispersal distances (2σ). Thus,

$$N_b = 4\pi D\sigma^2 \qquad \text{(Wright 1943)}$$

Rousset (1997, 2000) has developed methods for the estimation of neighbourhood size from pairwise calculations of F_{ST} between demes and between individuals. Typically, the method involves the calculation of pairwise genetic distances, and subsequent plotting against the natural logarithm of distance. The inverse of the slope of the regression provides an estimate of neighbourhood size (Rousset 1997, 2000).

Another summary statistic of importance is the calculation of θ, the composite estimate of population size (N) and mutation rate (μ); $2N\mu$ in haploids and $4N\mu$ in diploids. The need for estimating θ arises from the fact that neither of the two measures that comprise the parameter can be estimated independently from population genetic or sequence data without prior information on mutation rates or effective population size. Mutation rate, in particular is notoriously difficult to estimate from genetic data due to the occurrence of back mutations. Back mutations occur in DNA sequence data since mutation rate variation typically follows a distribution with few sites of high mutation rate and many sites of low mutation rates (Yang 1996). Several estimators of θ are evident in the literature: the expected number of alleles in an infinite-allele model (Ewens 1972), the number of segregating sites in an infinite-site model (Watterson 1975) or nucleotide diversity, the mean number of pairwise differences (Tajima 1983). The parameter θ also proves useful in maximum-likelihood inference of demographic parameters given the observed genetic data.

Furthermore, the relationship between different estimators of θ forms the basis for Tajima's D (Tajima 1989), the test of neutrality of mutations. Tajima's D is based on the premise that in a gene under selection, θ estimated from segregating sites will be substantially greater than θ estimated from nucleotide diversity, since rare mutations that are selected against are down-weighted in the calculation of the latter. Fu and Li (1993) have further developed neutrality tests based on the observation that purifying

selection is evident as an excess of mutations at the tips of a gene genealogy. Since these tests compare mutations in the recent past to mutations in the distant past, the use of an outgroup from a closely related species is recommended (Fu & Li 1993). Although neutrality tests have been criticized for lack of statistical power (Simonsen *et al.* 1995), a central problem relating to neutrality tests is rather the difficulty in distinguishing selection for a particular allele versus the demographic event of rapid population growth. Both of these processes generate the same genetic signal (Tajima 1989), evident as a star-like pattern in a gene tree, or unimodal distribution of pairwise genetic differences, or mismatch distributions (Rogers & Harpending 1992). The use of multiple loci is thus crucial in any population genetics study, where selection will be evident at only those loci under selection, whereas demographic histories will be evident in the genetic patterns of all loci.

An extension to the analysis of mismatch distributions is the development of methods to analyse a spatial range expansion versus that of simply an increase in population size (Ray *et al.* 2003, Excoffier 2004). Initially, a coalescent simulation combined with a demographic model of spatial expansion was used to observe the effects of spatial expansions on intra-deme molecular diversity in a structured population model (Ray *et al.* 2003). The results in general indicated that under low levels of migration between demes a spatial demographic expansion generated multimodal mismatch distributions. In contrast, large levels of migration between demes generated a pattern that was indistinguishable from a structured population that had always been exchanging a large number of migrants (Ray *et al.* 2003). Excoffier (2004) further utilized these simulation results to derive an analytical expression of F_{ST} given a structured population that has undergone a recent spatial demographic expansion. The process of simulating a demographic process and observing the effects on patterns of genetic diversity and subsequently devising an analytical algorithm is conducive to the advancement of population genetic theory. In addition, the results from a simulation model, and the data generated, can be subsequently input into the analytical algorithm such that its power to detect demographic processes can be evaluated.

Model-based approaches

Model-based approaches to analyzing and interpreting spatial genetic patterns are subdivided into three general approaches, comparative simulation modeling, analytical-based inference and simulation-model-based inference. As is evident in the example above (Ray *et al.* 2003) comparative simulation modeling provides a means to understand the effects that a particular demographic process may have on inference using summary statistics. Typically, comparative simulation modeling would comprise the repeated simulation of a demographic event, and the repeated estimation of a summary statistic of interest. General conclusions can thus be drawn regarding the effect of the demographic event on the calculation of the statistic of interest. Such simulations can either be effected forward-through-time where every individual in the population is simulated, and a sub-sample is drawn for the calculation of summary statistics; or backwards-through-time where only the genealogical history of the sampled alleles need to be simulated. Such backwards-through-time models, based on coalescent theory, have become popular due to their simplicity and mathematical tractability. The neutral coalescent, based on a panmictic Wright-Fisher population of constant population size, with no selection and no recombination, simply states that of all possible events that could happen to a sample of *n* alleles one generation back in time, only two are important: either all *n* alleles have distinct parents, or two alleles in the sample share a common ancestor (Wakeley 2004). This process continues from the samples observed in the present, backwards through time, until all samples (lineages) have coalesced to a single common ancestor. The result is a genealogy of the samples characterized by *n-1* coalescent events, and a distribution of times to each coalescent event (branch lengths). Kingman (1982) showed that the probability that *n* alleles are reduced to *n-1* alleles in the previous generation, in a total population size *N*, is given by

$$P_n = \frac{n(n-1)}{4N} \qquad \text{Kingman (1982)}$$

and thus the estimated time for *n* alleles to be reduced to *n-1* alleles is given by

$$E(T_n) = \frac{4N}{n(n-1)} \qquad \text{Kingman (1982)}$$

Therefore, since the genealogy comprises *n-1* coalescent events the total time to coalescence is given by

$$E(t) = \sum_{i=2}^{n} E(T_i) = 4N(1 - \frac{1}{n})$$

Per generation coalescent demographic simulation is a two-step process. Firstly, a random genealogy is simulated under the demographic model of interest, followed by the scattering of mutations onto the genealogy given the branch lengths and a mutation rate. The demographic model of population size fluctuations for example, could be implemented through varying *N* in the equations above at each generation to calculate the probability of coalescence. These probabilities are used to construct random genealogies, by drawing a random number at each generation. Coalescent events occur when a random number is greater than the probability of coalescence at the particular generation and is subsequently recorded as a coalescent event in the genealogy. Mutations are scattered forward through time on each simulated genealogy according to the branch lengths and mutation rates, and thus genetic data is generated at the tips of the genealogy. The process is repeated thousands of times, each time generating data and calculating a summary statistic of interest. Distributions of the summary statistics of interest can then be observed to infer the effect of the demographic process simulated. The result is an understanding of the potential variance in genetic data under a particular demographic model, given that the gene sorting process within populations is stochastic. Leblois *et al.* (2004) have used such an approach to determine the effects of temporal changes in density and dispersal on the inference of neighbourhood size in continuous populations.

In some cases analytical results are available for statistics of interest. For example, the analytical formulation of the number of segregating sites in a sample *n* can be used to make a maximum likelihood estimate of θ. However, this represents a point estimate of θ, whereas a degree of error is often required. Obtaining a variance for such estimation is not yet possible (Wakeley 2004). Furthermore, the variance of an estimator is only useful when the errors are normally distributed, or when the distribution is known and symmetric, a characteristic atypical of genetic data

(Wakeley 2004). Thus the analytical-based inference of demographic parameters given genetic data is limited and a simulation-based model of inference is required.

Simulation-based inference is dependent on maximum-likelihood (Edwards 1972) and Bayesian methods (Bayes 1763), and thus these are first introduced. Likelihood-based inference has the aim of evaluating the likelihood of a particular parameter given the observed data. Thus the calculation of the probability $P(D \mid \psi)$, of observing data, D, given the parameters, ψ, of a particular model is performed. The parameters of the model typically include the genealogy of the sampled alleles, the population size-mutation rate composite, θ, migration rates, and rates of population size increase or decrease. Bayesian inference procedures also utilize likelihood in the calculation of a posterior distribution for parameters, ψ, but further allow the incorporation of prior knowledge of the system under consideration. Such prior knowledge may take the form of a direct-estimate of migration rates between demes, or some prior belief of the effective population size. The posterior distribution is given by

$$P(\psi \mid D) = \frac{P(D \mid \psi)P(\psi)}{P(D)} \qquad \text{Bayes (1763), Edwards (1972)}$$

where $P(\psi)$ is the distribution of a prior. Priors in population genetic applications are typically uniform and given sufficient data the posterior distribution is dominated by the likelihood, such that the choice of prior has little effect on the conclusions drawn (Stephens 2003). In the following discussion of inference, however, I will concentrate on likelihood-based inference since the procedure of estimation in Bayesian inference is similar.

Coalescent-based likelihood inference is based on the distribution of genealogies, τ, given the observed data. Associated with each genealogy are parameters, such as branches, or mutations on the branches. Given that τ is known, one could calculate the probability of the data given the parameters of the model, i.e. $P(D \mid \tau, \psi)$. However, τ is unknown and as such one needs to calculate the likelihood of a particular parameter, or suite of parameters, by summing over all possible genealogies. This is performed since a single data set can be obtained from many different genealogical

histories. As such, the likelihood of the parameter is the sum of all probabilities for all potential genealogies, given the parameters, ψ

$$L(\psi) = P(D \mid \psi) = \sum_{\tau} P(D \mid \tau, \psi) P(\tau \mid \psi) \qquad \text{Felsenstein (1973, 1988)}$$

The number of possible genealogies increases rapidly with the number of samples, for three samples there are only three possible genealogies, for 10 samples 2571912000 genealogies and for 100 samples, 1.37×10^{284} genealogies (Felsenstein 2004). Thus, summing over all possible genealogies becomes computationally unfeasible given the sample sizes of typical population genetics studies. However, a glimmer of hope resides in the observation that the proportion of genealogies that contribute significantly to the sum across all genealogies is less than one in a million (Stephens 2003). This forms the basis of inference techniques, such as Monte Carlo integration (Hastings 1970), which have the purpose of estimating the likelihood surface through estimating the aforementioned likelihoods over parameter space. In summary, MCMC focuses the calculation of likelihoods in genealogy space where the genealogies that contribute most to the likelihood reside. Thus the amount of computation is reduced and likelihood surfaces can be approximated. Typically, the process involves starting at a particular point in genealogy space, evaluating the probabilities in the likelihood function above, proposing an alteration to the genealogy such that a move in genealogy space is suggested, and deciding whether to move to the suggested point based on the ratio of the current probability to that of the proposed point. These movements through genealogy space are typically referred to as chains. Coalescence and importance sampling are extensively reviewed in both Stephens (2003) and Felsenstein (2004) and thus will not be considered further.

Currently, the coalescent-based inference of demographic parameters is restricted to only a few migration models and demographic scenarios. These scenarios include the estimation of (i) gene flow in structured populations (Beerli & Felsenstein 1999, 2001), (ii) population growth/decline in both unstructured (Kuhner *et al.* 1998) and structured populations (Kuhner *et al.* 2004), (iii) divergence time of two populations that exchange/exchanged migrants (Nielsen & Wakeley 2001) and (iv) recombination rates in structured/unstructured populations (Fearnhead and Donnelly 2001, Kuhner *et al.* 2004). There is a potential to include more demographic scenarios in coalescent-

based models, such as fluctuating population size in structured populations, and indeed Knowles and Maddison (2002) and Knowles (2004) have noted that custom-development of species-specific models for demographic inference will be the future of phylogeographic analysis. Thus far I have only considered maximum likelihood and Bayesian inference models that make full use of the sequence or allelic data provided. However, there is a trend in the population genetics literature towards maximum-likelihood or Bayesian inference conditional on summary statistics (Tavaré *et al.* 1997, Weiss & von Haeseler 1998, Beaumont *et al.* 2002). These methods differ from the inference process described above in that at each step in the chain, the probability of the data given the parameters $P(D|\psi)$ is replaced with the probability of some summary statistic, k, given the parameters, i.e. $P(k|\psi)$. Given that the summary statistic is a sufficient representation of the data, the evaluation of these likelihoods is computationally faster than full-data methods. These methods may allow the development of multi-parameter custom likelihood models for genetic data analysis.

Dissertation outline

Chapter 1: Introduction and literature review

Chapter 2: Temporal and spatial distribution of African Wild Silk Moth, *Gonometa postica*, eruptions in southern Africa

This chapter presents three years of presence/absence distribution data. The potential cause of large-scale temporal changes in the distribution of eruptions is discussed and recommendations are made with regard to future temporal data collection, and climatic modeling.

Chapter 3: Characterisation of six microsatellite loci in the African Wild Silk Moth (*Gonometa postica*, Lasiocampidae)

Species-specific microsatellite markers were developed for *Gonometa postica* using an AFLP-based enrichment protocol (Zane *et al.* 2002). The six loci are reported along with estimates of Hardy-Weinberg and linkage-disequilibrium. This chapter is in press with *Molecular Ecology Notes*.

Chapter 4: The effect of large annual population size fluctuations on spatial genetic pattern in the continuously distributed African Wild Silk Moth (*Gonometa postica*)

The results of a single years analysis of microsatellite and mtDNA genetic data are presented in this chapter. The results are peculiar in that a species that has low dispersal ability and only lives for a few days appears to have little evidence for isolation by distance in the data. Simulations are subsequently used in an effort to understand the effects of population size fluctuations on spatial genetic pattern in a continuously distributed species. The analysis in this chapter falls within the section 'comparative simulation modeling' as described above.

Chapter 5: Temporal and spatial genetic patterns in the African Wild Silk Moth (*Gonometa postica*) and implications for cyclical population dynamics

The population genetic results from three years of successive sampling are presented in this chapter. The analysis is focused on estimating the degree of population size changes and detecting whether there may be weak signals of migration in the data. Standard maximum-likelihood techniques are used, and the implications of the assumptions of these techniques in analyzing the data are discussed.

Chapter 6: CoalFace: a graphical user interface program for the simulation of coalescence

Coalescent analysis is at the forefront of current population genetics, and thus in the preparation for a local workshop on phylogeographic analysis I developed a tool to teach the applications of coalescent theory. This chapter describes the software and its applicability in teaching population genetics and phylogeography.

Chapter 7: Conclusions

Appendix I: A population genetics pedigree perspective on the transmission of *Helicobacter pylori*

During my PhD I collaborated on a project to determine the mode of transmission of *Helicobacter pylori* using gene sequences and an extensive pedigree. The study presented a unique opportunity to investigate transmission of this bacterium, due to the abnormally high prevalence in a local South African community and the extensive

sampling. I developed a simulation model, in collaboration with co-authors, which contrasted different modes of transmission and their effect on observed genetic data. This chapter is included as an Appendix here since although the species, and question are different from my principal dissertation research, the method of using simulation modeling for inference is common.

Appendix II: LatticeFlucII source code

The C source code for the simulation model used in chapter 4 is presented.

Appendix III: CoalFace source code

The kylix/Delphi source code for the CoalFace program presented in chapter 6 is presented.

General Notes

Please note that all chapters (except 1 and 7) are written as manuscripts that have been or will be submitted for publication. Since all chapters have supervisors or collaborators as co-authors, I refer to the work being done by us, and not exclusively by myself. However, the research presented in this dissertation is entirely my own thought and execution, with useful discussions with my supervisors. The work presented in Appendix I, however, was a combined effort between Michael Cunningham, myself and other co-authors, and thus I have not included it in the main part of the dissertation. However, I still have first authorship for developing the simulation model and writing the manuscript. Since each chapter is in manuscript format, each has a list of authors as will appear in published form. In addition, the reference lists occur at the end of each chapter rather than at the end of the dissertation. Due to this format, there may be instances of duplication across chapters.

References

Akai H, Nakatomi R, Kioko E, Raina SK (1997) Fine structure of cocoon and cocoon filament from African *Gonometa* silkmoth (Lasiocampidae). *Int. J. Wild Silkmoth & Silk*, **3**, 15-22.

Avise JC (2000) Phylogeography: The history and formation of species. Harvard University Press, Harvard.

Avise JC, Arnold J, Ball RM, Bermingham E, Lamb T, Neigel JE, Reeb CA & Saunders NC. (1987) Intraspecific phylogeography: The mitochondrial DNA bridge between population genetics and systematics. *Annual Review of Ecology & Systematics*, **18**, 489-522.

Barton NH & Slatkin M (1986) A quasi-equilibrium theory of the distribution of rare alleles in a subdivided population. *Heredity*, **56**, 409-416.

Bayes T (1763) An essay towards solving a problem in the doctrine of chances. *Phil. Trans. Roy. Soc.*, **53**, 370-418. Reprinted In: Studies in the History of probability and statistics. IX. Thomas Bayes' essay towards solving a problem in the doctrine of chances. *Biometrika*, **45**, 293-315.

Beaumont MA, Zhang W, Balding DJ (2002) Approximate Bayesian Computation in Population Genetics, *Genetics*, **162**, 2025-2035.

Beerli P & Felsenstein J. (1999) Maximum-likelihood estimation of migration rates and effective population numbers in two populations using a coalescent approach. *Genetics*, **152**, 763-773.

Beerli P & Felsenstein J. (2001) Maximum-likelihood estimation of a migration matrix and effective population sizes in n subpopulations by using a coalescent approach. *Proceedings of the National Academy of Sciences, USA*, **98**, 4563-4568.

Bjornstad ON, Peltonen M, Liebhold AM & Balstensweiler W (2002) Waves of larch budmoth outbreaks in the European Alps. *Science*, **298**, 1020-1023.

Burton C, Krebs CJ, Taylor EB (2002) Population genetic structure of the cyclic snowshoe hare (*Lepus americanus*) in southwestern Yukon, Canada. *Molecular Ecology*, **11**, 1689-1701.

Crow JF (1986) *Basic Concepts in Population, Quantitative and Evolutionary Genetics*. WH Freeman, San Francisco.

Edwards ACF (1972) Likelihood: expanded edition. The Johns Hopkins University Press, London.

Ehrich D, Jorde PE, Krebs CJ, Kenney AJ, Stacy JE, Stenseth NC (2001) Spatial structure of lemming populations (*Dicrostonyx groenlandicus*) fluctuating in density. *Molecular Ecology*, **10**, 481-495.

Ewens WJ (1972) The sampling theory of selectively neutral alleles. *Theoretical Population Biology*, **3**, 87-112.

Excoffier L (2003) Analysis of population subdivision. In: Handbook of Statistical Genetics, 2nd Edition (eds. DJ Balding, M Bishop, C Cannings), pp 713-750, John Wiley & Sons, Ltd.

Excoffier L (2004) Patterns of DNA sequence diversity and genetic structure after a range expansion: lessons from the infinite-island model. *Molecular Ecology*, **13**, 853-864.

Fearnhead P & Donnelly P (2001) Estimating recombination rates from population genetic data. *Genetics*, **159**, 1299-1318.

Felsenstein J (1973) Maximum likelihood estimation of evolutionary trees from continuous characters. *American Journal of Human Genetics*, **25**, 471-492.

Felsenstein J (1988) Phylogenies from molecular sequences: inference and reliability. *Annual Review of Genetics*, **22**, 521-565.

Felsenstein J (2004) Inferring phylogenies. Sinauer Associates Inc. Massachusetts.

Finerty JP (1979) Cycles in Canadian Lynx. *American Naturalist*, **114**, 453-455.

Freddi G, Bianchi Svilokos A, Ishikawa H, Tsukada M (1993) Chemical composition and physical properties of *Gonometa rufobrunnea* silk. *Journal of Applied Polymer Science*, **48**, 99-106.

Fu Y-X & Li W-H (1993) Statistical tests of Neutrality of Mutations. *Genetics*, **133**, 693-709.

Fuller SJ, Wilson JC & Mather PB (1997) Patterns of differentiation among wild rabbit populations *Oryctolagus cuniculus* L. in arid and semiarid ecosystems of north-eastern Australia. *Molecular Ecology*, **6**, 145-153.

Gaggiotti OE (1996) Population genetic models of source-sink metapopulations. *Theoretical Population Biology*, **50**, 178-208.

Hartland-Rowe R (1992) The Biology of the wild silkmoth *Gonometa rufobrunnea* Aurivillius (Lasiocampidae) in northeastern Botswana, with some comments

on its potential as a source of wild silk. *Botswana Notes and Records*, **24**, 123-133.

Hastings WK (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**, 97-109.

Ibrahim KM (2001) Plague dynamics and population genetics of the desert locust: can turnover during recession maintain population genetic structure. *Molecular Ecology*, **10**. 581-591.

Ibrahim KM, Nichols RA, Hewitt GM (1996) Spatial patterns of genetic variation generated by different forms of dispersal during range expansion. *Heredity*, **77**, 282-291.

Kimura M & Weiss GH (1964) The stepping stone model of population structure and the decrease of genetic correlation with distance. *Genetics*, **61**, 763-771.

Kingman JFC (1982) The coalescent. *Stochastic Processes and their Applications*, **13**, 235-248.

Knowles LL, Maddison WP (2002) Statistical phylogeography. *Molecular Ecology*, **11**, 2623-2635.

Knowles LL (2004) The burgeoning field of statistical phylogeography. *Journal of Evolutionary Biology*, **17**, 1-10.

Kuhner MJ, Yamato J, Felsenstein J (1998) Maximum likelihood estimation of population growth rates based on coalescent. *Genetics*, **149**, 429-434.

Kuhner MK, Yamato J, Beerli P, Smith LP, Rynes E, Walkup E, Li C, Sloan J, Colacurcio P, Felsenstein J (2004) LAMARC v 1.2.1. University of Washington, http://evolution.gs.washington.edu/lamarc.html.

Leblois R, Rousset & Estoup A. (2004) Influence of spatial and temporal heterogeneities on the estimation of demographic parameters in a continuous population using individual microsatellite data. *Genetics*, **166**, 1081-1092.

Nagylaki T (1982) Geographical invariance in population genetics. *Journal of Theoretical Biology*, **99**, 159-172.

Nei M (1987) Molecular Evolutionary Genetics. Columbia University Press, New York, NY, USA.

Nielsen R, Wakeley JW (2001) Distinguishing Migration from Isolation: an MCMC Approach. *Genetics*, **158**, 885-896.

Ray N, Currat M, Excoffier L (2003) Intra-deme molecular diversity in spatially expanding populations. *Molecular Biology and Evolution*, **20**, 76-86.

Rogers AR, Harpending H (1992) Population growth makes waves in the distribution of pairwise genetic differences. *Molecular Biology and Evolution*, **9**, 552-569.

Rousset F (1997) Genetic differentiation and estimation of gene flow from F-statistics under isolation by distance. *Genetics*,**145**, 1219-1228.

Rousset F (2000) Genetic differentiation between individuals. *Journal of Evolutionary Biology*, **13**, 58-62.

Simonsen KL, Churchill GA, Aquadro CF (1995) Properties of statistical tests of neutrality for DNA polymorphism data. *Genetics*, **141**, 413-429.

Slatkin M (1985) Rare alleles as indicators of gene flow. *Evolution*, **39**, 53-65.

Slatkin M (1987) Gene flow and the geographic structure of natural populations, *Science*, **236**, 787-792.

Smit N (1999) A Guide to the Acacias of South Africa. Briza, Pretoria.

Stephens M. (2003) Inference under the coalescent. In: *Handbook of Statistical Genetics, 2nd Edition.* (eds. DJ Balding, M Bishop, C Cannings), pp 636-661, John Wiley & Sons, Ltd.

Tajima F (1983) Evolutionary relationships of DNA sequences in finite populations. *Genetics*, **105**, 437-460.

Tajima F (1989) Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics*, **123**, 585-595.

Tallmon DA, Draheim HM, Mills L & Allendorf FW (2002) Insights into recently fragmented vole populations from combined genetic and demographic data. *Molecular Ecology*, **11**, 699-709.

Tavaré S, Balding DJ, Griffiths RC, Donnelly P (1997) Inferring coalescence times from DNA sequence data. *Genetics*, **145**, 505-518.

Turchin (2003) Complex Population Dynamics: a theoretical/empirical synthesis. *Monographs in Population Biology*, **35**. Princeton University Press, Princeton.

Vandewoestine S, Neve G, Baguette (1999) Spatial and temporal population genetic structure of the butterfly Aglais urticae L. (Lepidoptera, Nymphalidae). *Molecular Ecology*, **8**, 1539-1543.

Veldtman R (2004) The ecology of southern African wild silk moths (*Gonometa* species, Lepidoptera: Lasiocampidae): consequences for their sustainable use. University of Pretoria PhD thesis.

Wade MJ & McCauley DE (1988) Extinction and recolonisation: their effects on the genetic differentiation of local populations. *Evolution*, **42**, 995-1005.

Wakeley J (2004) Metapopulations and coalescent theory. In: *Ecology, Genetics and Evolution of Metapopulations* (eds. I. Hanski & O. E. Gaggiotti), pp 175-198. Elsevier Academic Press, London.

Watterson G (1975) On the number of segregating sites in genetical models without recombination. *Theoretical Population Biology*, **7**, 256-276.

Weir BS, Cockerham CC (1984) Estimating F-statistics for the analysis of population structure. *Evolution*, **38**, 1358-1370.

Weir BS, Hill WG (2002) Estimating F-statistics. *Annual Review of Genetics*, **36**, 721-750.

Weiss G, von Haeseler (1998) Inference of population history using a likelihood approach. *Genetics*, 149, 1539-1546.

Whitlock MC & McCauley (1990) Some population genetic consequences of colony formation and extinction: genetic correlations with founding groups. *Evolution*, **44**, 1717-1724.

Whitlock MC (2004) Selection and drift in metapopulations. In: *Ecology, Genetics and Evolution of Metapopulations* (eds. I. Hanski & O. E. Gaggiotti), pp 153-174. Elsevier Academic Press, London.

Wright S (1931) Evolution in Mendelian populations. *Genetics*, 16, 97-159.

Wright S (1940) Breeding structure of populations in relation to speciation. *American Naturalist*, **74**, 232-248.

Wright S (1943) Isolation by distance. *Genetics*, **28**, 114-138.

Wright S (1951) The genetical structure of populations. *Ann. Eugen.*, **15**, 323-354.

Yang Z (1996) Among-site rate variation and its impact on phylogenetic analysis. *Trends in Ecology and Evolution*, **11**, 367-372.

Chapter 2

Temporal and spatial distribution of African Wild Silk Moth, *Gonometa postica*, eruptions in southern Africa

"In considering the distribution of organic beings over the face of the globe, the first great fact which strikes us is, that neither the similarity nor the dissimilarity of the inhabitants of various regions can be wholly accounted for by climatal and other physical conditions"

Charles Darwin (1859)

Temporal and spatial distribution of African Wild Silk Moth, *Gonometa postica*, eruptions in southern Africa

Wayne Delport[1], Paulette Bloomer[1] & J. Willem H. Ferguson[2]

[1]Molecular Ecology and Evolution Programme, Department of Genetics, University of Pretoria, Pretoria, 0002, South Africa

[2]Centre for Environmental Studies, Department of Zoology and Entomology, University of Pretoria, Pretoria, 0002, South Africa

Abstract

The African Wild Silk Moth (*Gonometa postica*) experiences large inter-annual population size fluctuations in southern Africa. This species is currently of economic interest, as the focal species in the initiation of an African Wild Silk Industry. However, a consistent problem in the industry is the lack of a sufficient supply of cocoons. Therefore, we present data from three years of distribution surveys of eruptions of African Wild Silk Moths in southern Africa. Eruptions of the species show large degrees of variation on both spatial and temporal scales. In addition, we show an overall decrease in the persistence of eruptions over the sampling period. Furthermore, we use generalised additive modelling (GAM) to identify climatic variables that are significantly correlated with presence/absence of the species. Temperature and rainfall before the emergence of adult moths are significantly correlated with presence/absence. However, the effects of exogenous and endogenous factors on population cycles in this species are probably interlinked. Thus only long-term multidisciplinary population dynamics research will be sufficient to address the factors responsible for the complex population cycle observed in this species.

Keywords: distribution, generalised additive model, cyclical species

Introduction

The African Wild Silk moths (*Gonometa* spp., Lasiocampidae) are species that are presently of particular economic interest in southern Africa. The Liberty Life International Wild Silk Workshop held in Pretoria (4-5 November 2002) bears testament to the level of interest both in southern Africa and abroad. Both *Gonometa postica* and *G. rufobrunnea*, two species of African Wild Silk moth native to southern Africa, have been shown to possess a silk fibre of exceptional quality (Freddi *et al.* 1993, Akai *et al.* 1997) and attempts have already been made at initiating an industry for the utilization of *Gonometa* cocoons in the production of African Wild Silk. Such an example is Shashe Silk Pty (Ltd) in northwest Botswana. Shashe silk plummeted due to an insufficient supply of cocoons for the production of silk. The failure of Shashe Silk was the result of an incomplete understanding of the biology of the species and secondly, poor business management (McGeoch 2002). Shashe Silk ignored the cyclical nature of the species, and the lack of eruptions resulted in an unbalanced expenditure versus crop value ratio. Currently, several small-scale cottage industries are utilizing *Gonometa* fibres to weave a variety of textile products. Although these industries have been successful in utilizing *Gonometa* cocoons, a consistent complaint is the lack of a sufficient supply of cocoons. To this end a joint venture between the North West Provincial Government and the Council for Scientific and Industrial Research, called Wild Silk Africa, has been established in Ganyesa, north of Vryburg, in order to aid in the collection of cocoons and provide local communities with a source of revenue. However, such an industry cannot be based in a single region due to the cyclical and patchy nature of the population eruptions.

The *Gonometa* species in southern Africa have been shown to exhibit large inter-annual population fluctuations (Veldtman 2005). However, it is uncertain whether eruptions are only the result of local populations experiencing ideal conditions or whether current eruptions are initiated by dispersal of individuals from eruptive populations in previous generations. A second observation, regarding eruptions, is that they are patchily distributed at both the local and regional scale (Veldtman 2005). *G. postica* has a complex population cycle that is characterized by two generations per annum, the first starting in September-October when adult moths emerge from cocoons. Adults lack feeding mouthparts and survive for three to five days (Hartland-

Rowe 1992), during which breeding occurs. Eggs are laid, larvae emerge and pass through six instar larval stages in approximately five weeks, after which the larvae construct cocoons, pupate and enter a period of diapause. This period of diapause either carries through to the following September when adults emerge, or is broken in February with adult emergence and an additional population cycle. Typically, this second generation comprises between 12-50% of the first generation (Hartland-Rowe 1992), and culminates in pupae that emerge as adults in September. The contribution of this second cycle to eruptions of the species is unknown, and is necessary to consider in relation to other endogenous and exogenous factors.

In particular, the influence of exogenous factors, such as climate, on eruptions is necessary to understand before it can be determined whether cocoons are a sustainable resource for the African Wild Silk industry. Since it has been suggested that the patchy nature of *Gonometa* eruptions may be the result of spatial correlation in climatic variables (Veldtman 2005), we investigated the temporal correlation of rainfall and persistence of eruptions. Two main hypotheses, relating to climate, have been suggested. Firstly, some have noted the incidence of eruptions following periods of drought (Hartland-Rowe 1992). A putative explanation for this observation is the reduction of numbers of dipteran and hymenopteran larval parasites. Supposedly, a reduction in parasite numbers due to unfavorable conditions allows *Gonometa* populations to reach eruptive proportions. Secondly, large population crashes may be the result of heavy rainfall that causes high early instar mortality (Hartland-Rowe 1992). This hypothesis is supported by observations of low population sizes at sites that had experienced exceptionally heavy summer rainfall (Veldtman 2005). The purpose of this manuscript is to report on temporal variation in presence/absence of *G. postica*, and to address the influence of climate, in particular rainfall and temperature, on the distribution of eruptions over a three-year sampling period. Our results indicate a decline in eruptions over the sampling period, and furthermore statistical correlation of rainfall and temperature at the time of moth emergence with presence/absence.

Methods and materials

Distribution data collection

The purpose of this study was to assess the distribution of first generation *G. postica* eruptions across southern Africa for a period of three years (2002-2004). *G. postica*

experiences a period of suspended development or diapause during the winter months in southern Africa (May – August). Therefore, we surveyed the Kalahari region of southern Africa (Figure 1) on an annual basis (2002-2004) for the presence/absence of *G. postica* cocoons on *Acacia erioloba*, *A. tortilis*, *A. mellifera* and *A. haematoxylon* trees. We drove approximately 10 000km along roads in the Kalahari, stopping every 50km and checking for the presence or absence of cocoons. Although we only stopped every 50km, it is easy to spot cocoons whilst driving due to their size and conspicuousness. The larvae of *G. postica* are moderately polyphagous (Veldtman 2005), feeding on the leaves of two angiosperm families (Mimosaceae and Caesalpiniaceae). Although, we only intended to check for presence of cocoons on the *Acacia* species, it was nonetheless easy to spot cocoons on other species, such as the alien *Prosopis glandulosa*. Both the presence and absence of cocoons were recorded along with GPS reading of each locality, for each of the survey years. We drove the same route in 2002 and 2004, and a shorter route in 2003. Therefore, temporal changes in the presence/absence of eruptions do not represent un-sampled localities, but rather a true change in presence/absence of eruptions.

Climate data collection

Climate data, including total monthly rainfall and average monthly temperature, for the period 2001-2004 were obtained from the South African Weather Service. Data from a total of 167 rainfall and 35 temperature stations, distributed across the Northern Cape and North West provinces (Figure 1), were utilised in subsequent analysis. However, since these climate stations did not overlap with distribution presence/absence sampling localities, it was necessary to construct climate surfaces using interpolation techniques. We used an Inverse Distance Weighted interpolation procedure (power = 2, search radius = variable, number of points = 12) implemented in the spatial analyst extension of ArcGis 9 (www.esri.com). Thereafter, rainfall and temperature values were calculated, from these interpolated surfaces, for each of the presence/absence distribution localities surveyed, in each of the sampling years (2002 – 2004). Climatic variables (predictors) used in the distribution modelling were chosen so as to be informative with regard to the species' life cycle. The principal host species, *A. erioloba*, experiences a leaf flush in September. The timing of bud-burst is critical for many insects, with many insects exhibiting synchrony of egg laying and host-plant bud-burst (Dixon 2003). Therefore, the relationship between climatic factors and host-plant leaf flush, and the synchrony of egg laying by female

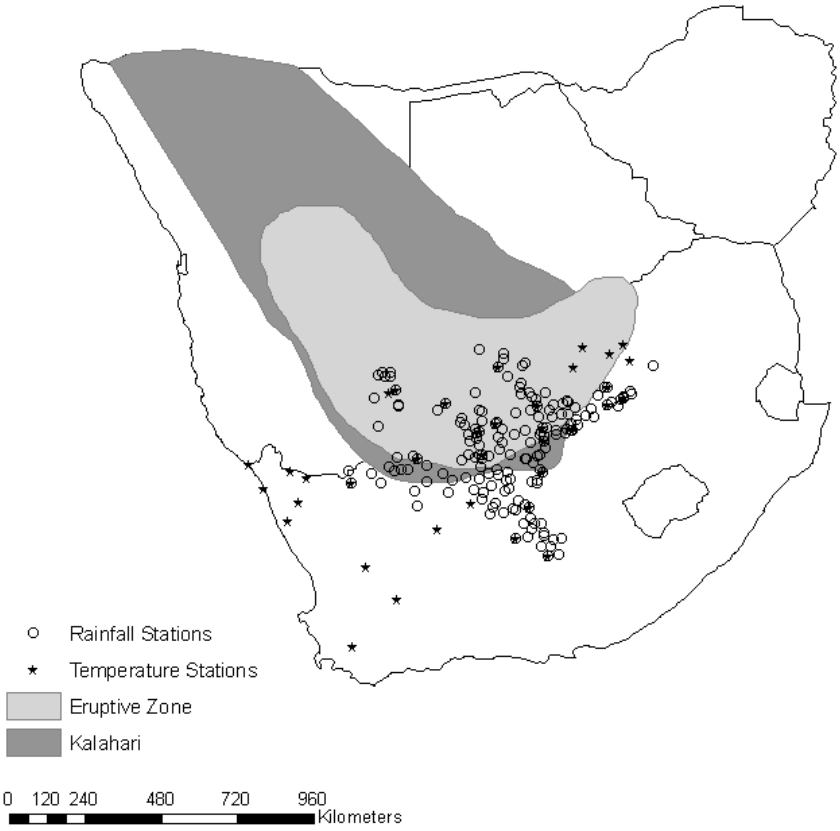


Figure 1: Distribution of rainfall and temperature weather stations used for the calculation of interpolated climate surfaces. The extent of the eruptive zone and distribution of the Kalahari biome in southern Africa is also shown.

G. postica moths may be important in the persistence of local populations. Climatic variables that were considered of importance for this study were (i) pre-emergence rainfall and temperature, (ii) rainfall and temperature during larval development, and (iii) rainfall and temperature coinciding with the second generation (Table 1). As a result of the long pupal diapause, presence/absence response variables from any year is directly related to the larval development and pupation success of the previous year. These variables were therefore used as predictors of species distribution in the subsequent predictive distribution modelling (Table 1). All ArcGis 9 analysis utilized the GCS_Hartebeeshoek_1994 geographic coordinate system.

Distribution modelling

The purpose of the distribution modelling was to determine which of the climatic predictor variables were correlated with presence/absence data (i.e. response variables) collected from 2002-2004. Although it is not possible to determine causation from correlation (Sokal & Rohlf 1981), this approach still allows a statistical description of functional relationships between environmental and response variables. We used a generalized additive model (GAM), which is a special semi-parametric case of the generalized linear model (GLM), that allows regressions to be estimated for non-standard distributions (Poisson, binomial, gamma), and allows better fitting of species response curves to environmental gradients (Austin 2002). Specifically, we use the GRASP package (Lehmann *et al.* 2002) implemented in the R statistical language (R Development Core Team 2005). The modelling approach was subdivided into three steps; (i) data exploration, (ii) model selection, and (iii) model interpretation. Firstly, exploratory analyses were conducted to describe the environmental space occupied by the species, and to detect statistical correlations between variables. In the former we plotted histograms of distribution data with respect to the environmental variables they represent. Correlated predictor variables may be problematic in the estimation of additive surfaces (Lehmann *et al.* 2002) and therefore we identified variables that were strongly correlated. Secondly, model selection was performed, under a quasi-binomial model, through a stepwise procedure that selects significant predictor variables. The process begins with a starting model, and then eliminates climatic predictors on the basis of whether the predictor provides a significantly better fit (ANOVA) to the observed data than a model without the current predictor. Finally, we used response curves to ascertain how presence/absence is related to the environmental predictors.

Table 1: Climatic predictors and response variables used to model the distribution of *Gonometa postica* eruptions. JAS = July – August – September, OND = October – November – December, JFM = January – February – March. *Data unavailable for this study.

	Presence/absence		
	2002	**2003**	**2004**
Temperature (average monthly - °C)			
Pre-emergence (T_{pe})	JAS 2001	JAS 2002	JAS 2003
Larval development (T_l)	OND 2001	OND 2002	OND 2003
Second generation (T_2)	JFM 2002	JFM 2003	JFM 2004
Precipitation (total - mm)			
Pre-emergence (P_{pe})	JAS 2001	JAS 2002	JAS 2003
Larval development (P_l)	OND 2001	OND 2002	OND 2003
Second generation (P_2)	JFM 2002	JFM 2003	JFM 2004*

Results

Distribution results

The survey of first generation cocoons conducted in the July 2002 indicates the patchy nature of eruptive populations in *G. postica* (Figure 2a). Sampling effort during 2003 (Figure 2b) was less intense and at a smaller geographic scale, yet the intensity at this reduced scale was equivalent to that of 2002. A general observation from 2003 distribution data is the observation of clusters of presence and absence data (Figure 2b). Furthermore, comparatively fewer presence localities were found versus that in 2002 (Table 2). The reduction in the number of eruptions in 2004 is even more severe (Table 2, Figure 2c) with even fewer presence localities recorded, and sampling effort equivalent to that of the 2002 survey.

Modelling results

Exploratory analysis of the predictor and response variables indicated a low incidence of presence samples (Figure 3, Table 2). Presence samples are, however, clumped in the distribution of pre-emergence temperature (Figure 3a). Correlations between variables indicate that most variables are correlated, as one would expect for climate data. However, no two variables have *r*, the correlation coefficient, greater than 0.80. Therefore, all variables were retained for the model selection process. Stepwise selection of statistically significant climatic predictors for the distribution of *G. postica* identified the following model:

$$Y = s(T_{pe}, 2.605) + s(T_2, 1.009) + s(P_{pe}, 4.391)$$

where Y = response variable, i.e. presence/absence, s = smoothing coefficient, and predictor variables are as in Table 1. Two variables, temperature (T_l) and precipitation during larval development (P_l) were not selected as contributing significantly in the model selection procedure. Finally, response curves of three predictor variables included in the model indicated the relationships between predictor and response variables (Figure 5), i.e. the species distribution. Response curves for temperature and precipitation before the emergence of adult moths are complex and clearly not linear relationships. Indeed, precipitation before emergence (P_{pe}) indicates that intermediate total precipitation over the three-month period (July – September) is likely to result in fewer eruptions. However, both low and higher levels of precipitation at this time

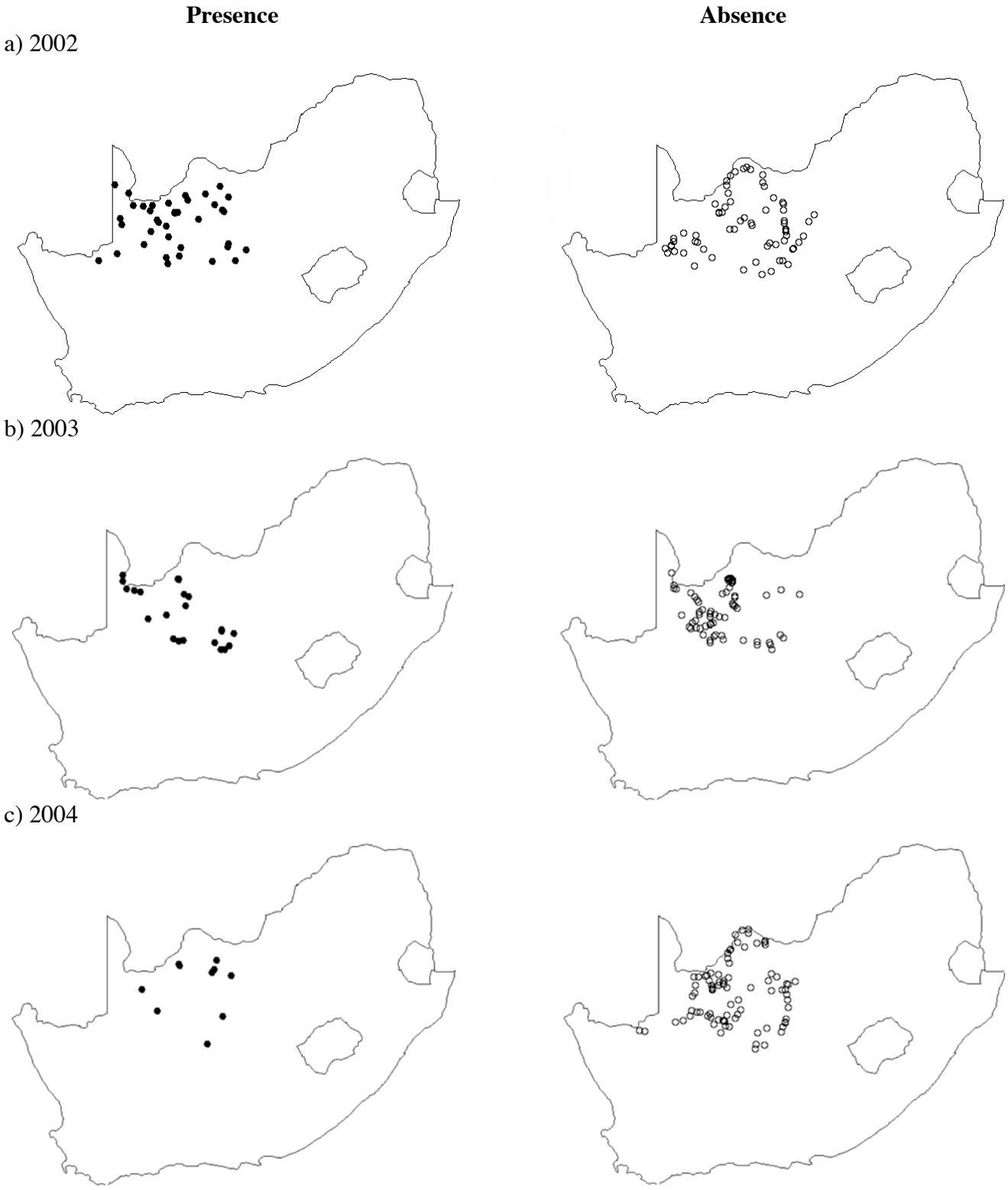


Figure 2: Temporal change in presence and absence of *G. postica* eruptions as determined with annual surveys from 2002-2004.

Table 2: Total number of sampled sites and proportion of presence sites of all sites sampled during 2002, 2003 and 2004.

Sampling year	Total sites sampled	Proportion presence
2002	185	0.42
2003	98	0.25
2004	200	0.07

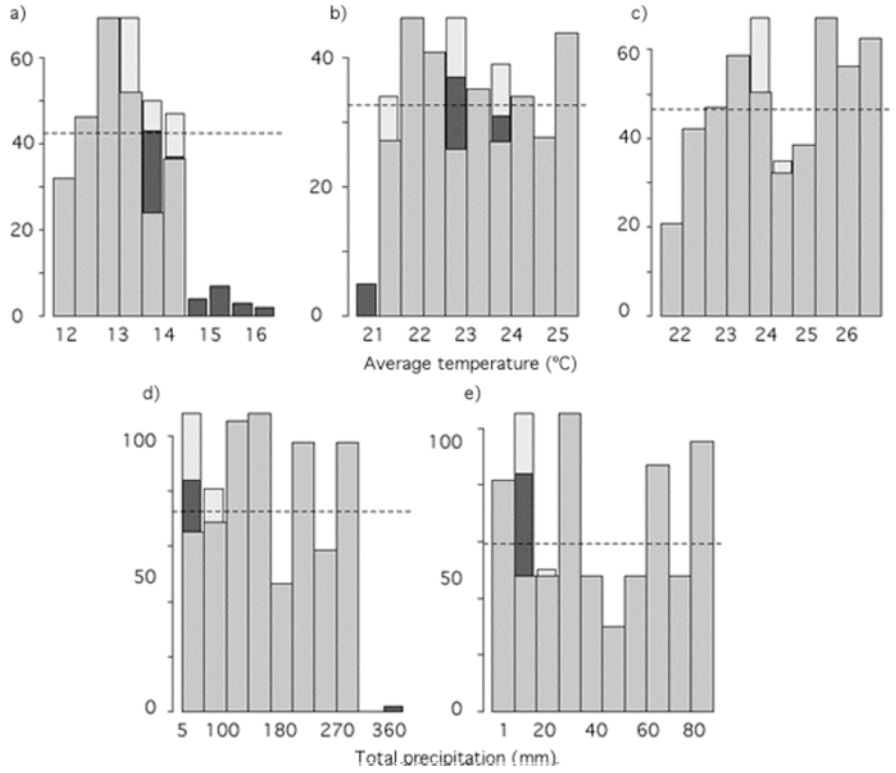


Figure 3: Distribution of environmental predictors and the occurrence of presence/absence samples. Histogram bars represent the distribution of both presence and absence samples, where dark areas are the distribution of presence samples. a) T_{pe} - Pre-emergence temperature, b) T_1 - larval development temperature, c) T_2 - Second generation temperature, d) P_{pe} - Pre-emergence precipitation, e) P_1 - larval development precipitation.

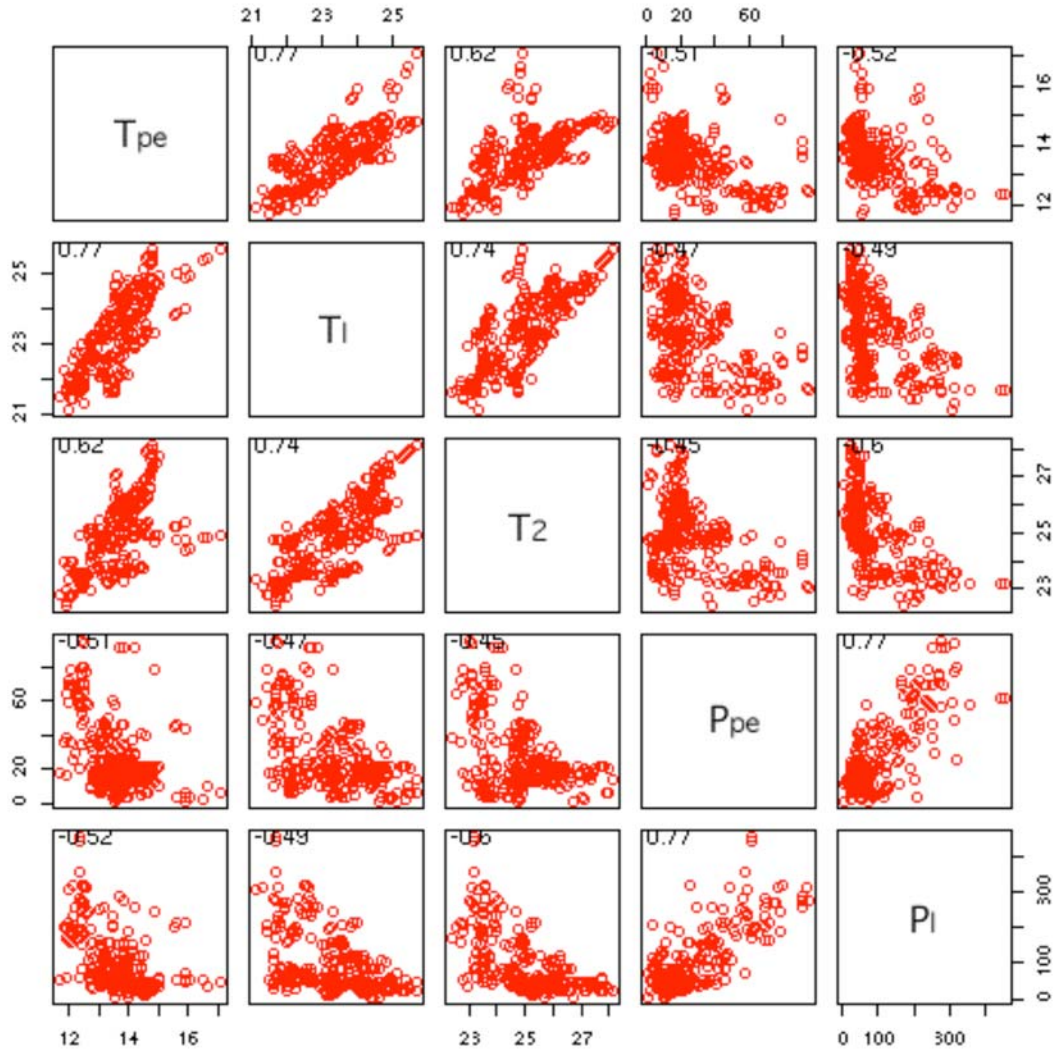


Figure 4: Statistical correlations between pairs of climate predictor variables used in the GAM model. Climate variables are named as in Table 1.

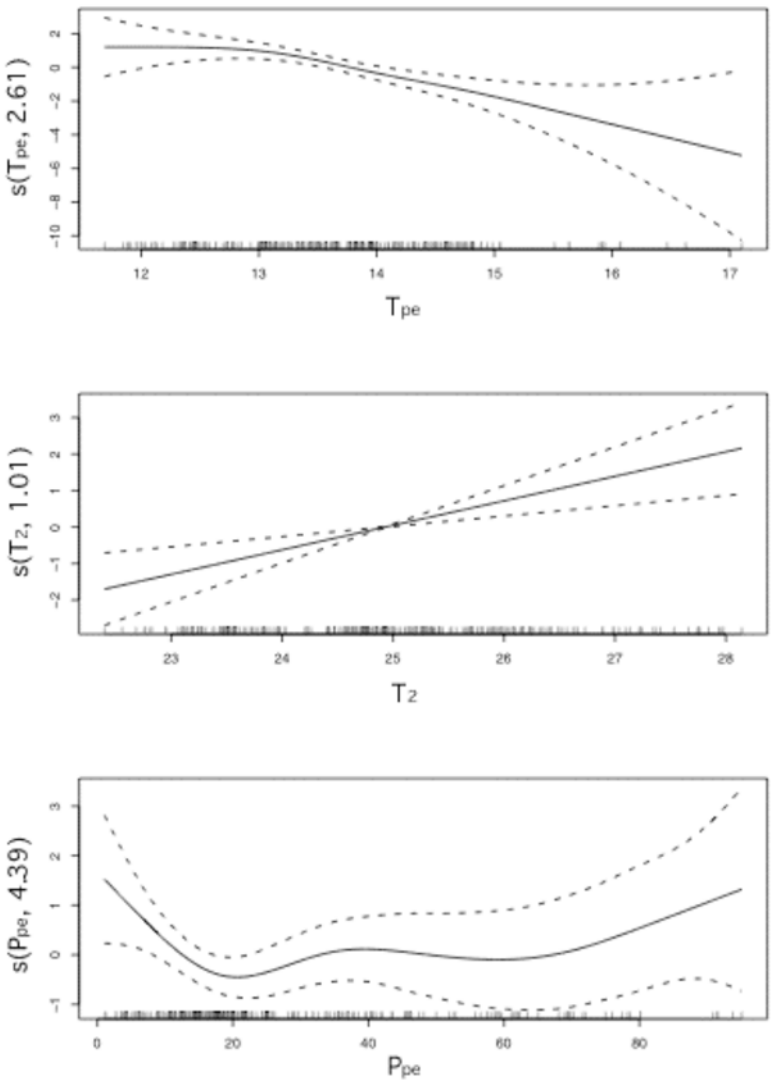


Figure 5: Response curves of predictor variables in selected generalized additive model (GAM). In each case the relationship between the predictor variable and its implementation in the model is represented. Predictor variables are named as in Table 1.

seem to be correlated with presence of the species. Lower temperatures before emergence are correlated with presence of the species (Figure 5), yet response curves for temperature during the second generation suggests the inverse of this relationship. Although, some of the results seem contradictory, the current model does provide a framework for future work and discussion.

Discussion

The distribution of *G. postica* clearly shows a pattern characteristic of an eruptive species, with little temporal consistency in population eruptions. Of particular interest in this study are (i) the temporal decline in *G. postica* eruptions, (ii) the degree of patchiness and the persistence of these patches, and (iii) the correlation of presence/absence with environmental variables. Firstly, we observed an overall decline in the presence of *G. postica* cocoons over the three years sampled in this study. Although, sampling effort was not consistent over the three years, with less sampling in 2003, these results are still evident in the proportion of presence localities observed relative to the total number of localities sampled (Table 2). We found pupae in approximately 42% of sampled localities in 2002, yet only 7% of the sites sampled in 2004 had pupae present. This represents a considerable decrease in population size, and highlights the unpredictable nature of this species. The degree of spatial and temporal heterogeneity in this species is severe. These results are supported by previous work, where temporal *G. postica* pupal abundance at local sites was shown to range across two orders of magnitude (Veldtman 2005). However, some areas do appear to have eruptions that have persisted over time (Figure 2). The correlation of these clusters of presence data with environmental variables was explored using generalized additive modelling (GAM). The selected model included precipitation and temperature before the emergence of adult moths as important determinants of presence/absence. These results are intuitive, since it is thought that temperature acts as a cue for the termination of diapause in this species (Hartland-Rowe 1992), and secondly, since total precipitation at this time may be important in determining foliage quality of the host species, at the time of its leaf flush. The results presented here, however, do not suggest causal relationships, but are merely correlations. Therefore, it would be necessary to conduct further studies investigating the relationships between these climatic variables and both silk moth population dynamics and host plant phenology. Although this model needs to be tested with additional data and validated

using cross-validation procedures (Fielding & Bell 1997), we do believe the approach followed here will assist in understanding the cyclical population dynamics of this species.

Several insect species have such population cycles characterized by large eruptions and population crashes (Turchin 2003). Examples that have been well-studied include the Larch budmoth, *Zeiraphera diniana*, (Bjornstad *et al.* 2002), southern Pine beetle, *Dendroctonus frontalis*, (Reeve *et al.* 1995), the desert locust, *Schistocerca gregaria* (Cheke & Holt 1993) and locally the armoured bush cricket, *Acanthoplus discoidalis*, (Holt *et al.* 2003, Minja & Green 2003. Turchin (2003) notes that the dynamics of such complex population cycles are attributed to the interaction between endogenous and exogenous factors. These factors encompass environmental effects, including climatic factors, population-specific effects such as density dependence and inter-species interactions, such as predator-prey relations. The overall decline in population eruptions of *G. postica* over the three sampling years in our study suggest that broad-scale spatial synchrony in population dynamics is evident in this species. Since dispersal, interspecific interactions (parasitism) and environmental variables are likely to influence the degree of spatial synchrony (Peltonen *et al.* 2002), it is difficult to determine whether the observed patterns are truly environmental. For instance broad scale patterns in rainfall might increase spatial synchrony, yet heterogeneity in the distribution of larval parasites might decrease spatial synchrony. Indeed Veldtman (2005) has detected some degree of spatial heterogeneity in the incidence of larval parasitism of *G. postica* at the local scale (100 trees), yet the inferred degree of heterogeneity was largely dependent on the spatial method used to quantify it. Spatial heterogeneity in larval parasites, and the influence of broad-scale climatic factors would tend to decrease and increase the degree of spatial synchrony in abundance, respectively, and thus complicate the inference of the effects of either factor in isolation. In addition, dispersal ability is likely to determine the scale at which spatial heterogeneity in abundance occurs, since adult mated female dispersal can seed the eruptions of subsequent years.

Therefore, it is imperative that several local *G. postica* populations should be monitored on an annual basis. Such local monitoring would include statistically rigorous estimates of abundance as suggested by Veldtman (2004), the collection of temporal life-history data, the direct estimation of dispersal ability with capture-mark-

recapture techniques, and the delineation of energetic requirements, in terms of foliage quality and quantity, of larvae to complete development. At the regional scale we propose the continued survey of both presence and absence distribution data, combined with abundance estimates at each site. These data combined with regional climatic information will allow the identification of climatic influences of *G. postica* eruptions. Given that some insect population cycles can occur on the order of 5-10 years (Turchin 2003) it is of utmost importance that these data are collected in a standardized manner over a long-time period. With the use of such long-term data we look forward to the development of population dynamics models that will allow the understanding of complex cycles in this species, and potentially enable the prediction of eruptions.

Acknowledgements

This work was funded by the Mellon Foundation Grant to J. W. H. Ferguson, P. Bloomer and W. Delport, and by a National Research Foundation grant to P. Bloomer (Gun: 2053653). The opinions and views presented in this article are however, not necessarily those of the National Research Foundation. Furthermore, we would like to thank Louis Hauman, Duncan McFadyen and E. O. Oppenheimer & Son for accommodation assistance during sampling.

References

Akai H, Nakatomi R, Kioko E, Raina SK (1997) Fine structure of cocoon and cocoon filament from African *Gonometa silkmoth* (Lasiocampidae). *Int. J. Wild Silkmoth & Silk*, **3**, 15-22.

Austin MP (2002) Spatial prediction of species distribution: an interface between ecological theory and statistical modelling. *Ecological Modelling*, **157**, 101-118.

Bjornstad ON, Peltonen M, Liebhold AM & Balstensweiler W (2002) Waves of larch budmoth outbreaks in the European Alps. *Science*, **298**, 1020-1023.

Cheke RA, Holt J (1993) Complex dynamics of Desert Locust plagues. *Ecological Entomology*, **18**, 109-115.

Dixon AFG (2003) Climate change and phonological asynchrony. *Ecological Entomology*, **28**, 380-381.

Fielding AH & Bell JF (1997) A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environmental Conservation*, **24**, 38-49.

Freddi G, Bianchi Svilokos A, Ishikawa H, Tsukada M (1993) Chemical composition and physical properties of *Gonometa rufobrunnea* silk. *Journal of Applied Polymer Science*, **48**, 99-106.

Hartland-Rowe R (1992) The Biology of the wild silkmoth *Gonometa rufobrunnea* Aurivillius (Lasiocampidae) in northeastern Botswana, with some comments on its potential as a source of wild silk. *Botswana Notes and Records*, **24**, 123-133.

Holt J, Mviha PJZ, Green SV (2003) Prediction of armoured bush cricket (Orthoptera: Tettigoniidae: Hetrodinae) outbreaks. Paper presented at the Symposium on IPM of Armoured Bush Crickets 6-9th July 2003 at the Congress of the Entomological Society of Southern Africa, University of Pretoria, South Africa.

Koch SO, Chown SL, Davis ALV, Endrödy-Younga, Van Jaarsveld AS (2000) Conservation strategies for poorly surveyed taxa: a dung beetle (Coleoptera: Scarabeidae) case study from southern Africa. *Journal of Insect Conservation*, **4**, 45-56.

McGeoch MA (2002) Liberty Life Trust Wild Silk Workshop: Summary document. 4-5 November, 2002. University of Pretoria.

Minja, EM, Green SV (2003) Armoured Bush Cricket Control - A Farmer Perspective. Paper presented at the Symposium on IPM of Armoured Bush Crickets 6-9th July 2003 at the Congress of the Entomological Society of Southern Africa, University of Pretoria, South Africa.

Peltonen M, Liebhold, Bjornstad ON, Williams W (2002) Spatial synchrony in forest insect outbreaks: roles of regional stochasticity and dispersal. *Ecology*, **83**, 3120-3129.

R Development Core Team (2005). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Reeve JD, Ayres MP, Lorio PL (1995) Host suitability, predation, and bark beetle population dynamics. In: Population dynamics: New approaches and synthesis (eds. Cappuccino N & Price P), pp 339-357. Academic Press, New York.

Sokal RR & Rohlf FJ (1995) Biometry: The principles and practice of statistics in biological research. W. H. Freeman and company, New York.

Turchin (2003) Complex Population Dynamics: a theoretical/empirical synthesis. *Monographs in Population Biology*, **35**. Princeton University Press, Princeton.

Veldtman R (2004) The ecology of southern African wild silk moths (*Gonometa* species, Lepidoptera: Lasiocampidae): consequences for their sustainable use. University of Pretoria PhD thesis.

Veldtman R, McGeoch MA & Scholtz CH (2002) Variability in cocoon size in southern African wild silk moths: implications for sustainable harvesting. *African Entomology*, **10**, 127-136.

Veldtman R, McGeoch MA & Scholtz CH (2004) The parasitoids of southern African wild silkmoths (Lepidoptera). *African Entomology*, **12**, 117-122.

Chapter 3

Characterisation of six microsatellite loci in the African Wild Silk Moth (*Gonometa postica*, Lasiocampidae)

"Have patience awhile; slanders are not long-lived. Truth is the child of time; erelong she shall appear to vindicate thee."

Immanuel Kant

Characterisation of six microsatellite loci in the African Wild Silk Moth (*Gonometa postica*, Lasiocampidae)

Wayne Delport[1], J. Willem H. Ferguson[2] & Paulette Bloomer[1]

[1]Molecular Ecology and Evolution Programme, Department of Genetics, University of Pretoria, Pretoria, 0002, South Africa

[2]Centre for Environmental Studies, Department of Zoology and Entomology, University of Pretoria, Pretoria, 0002, South Africa

Abstract

Six microsatellite markers were developed for the African Wild Silk Moth, *Gonometa postica* (Lasiocampidae) using an enrichment protocol. The total number of alleles ranged from 3 to 17 for a sample of 130 individuals across the distribution of the species. Observed levels of heterozygosity ranged from 0.17 to 0.78. Deviation from Hardy-Weinberg equilibrium detected in some loci is probably the result of large inter-annual population size fluctuations characteristic of this species. No evidence of linkage disequilibrium was detected among loci. These loci will be useful for the inference of demographic processes in a moth species that is of potential economic importance.

keywords: Lepidoptera, microsatellites, *Gonometa postica*, Lasiocampidae, FIASCO

Accepted for publication in *Molecular Ecology Notes*

The African Wild Silk Moth (*Gonometa postica*) is a species that could have considerable economic potential in rural southern Africa given that the larvae have been shown to yield a silk of high quality (Freddi *et al.* 1993). Indeed several small-scale cottage industries have been established using silk derived from these species. However, the potential for a rural-based silk industry utilising *G. postica* is largely dependent on annual population numbers. Given that *G. postica* shows large inter-annual fluctuations in population size the problem of insufficient supply of cocoons is evident. Therefore, population-based research is required such that an understanding of annual fluctuations in population size can be achieved. To this end we developed six microsatellite loci for the purpose of population genetic studies on *G. postica* in southern Africa.

Genomic DNA was extracted from wing muscle of one *G. postica* moth collected in the Northern Cape, South Africa. Thereafter a partial genomic library was constructed according to the FIASCO protocol (Zane *et al.* 2002). Approximately 50 ng of total genomic DNA was digested with *MseI* (New England Biolabs) and simultaneously ligated with T4 DNA Ligase (New England Biolabs) to *MseI* adaptors. The resulting fragments were PCR amplified with single-base 3' degenerate primers, and visualised with ethidium bromide staining under UV light on an agarose gel. Optimal PCR products were those that yielded a smear in the size range of 200-1000bp. The digested fragments were thereafter hybridised to 5' biotin labelled probes. Since it has proven difficult to isolate microsatellites from Lepidoptera (Meglecz *et al.* 2004) we used as many probes as feasible, yet combined non-complementary probes of the same motif length in single reactions. The following probes were used: $(gt)_{17}$, $(ca)_{15}$, $(ct)_8$, $(gc)_8$, $(tgc)_7$, $(cag)_5$, $(cca)_5$, $(cat)_5$, $(cac)_7$, $(ata)_8$, $(gtg)_5$, $(caa)_5$, $(aca)_5$, $(cga)_5$, $(cgca)_6$, $(tcca)_6$, $(tgtc)_6$, $(gata)_6$, $(tatc)_6$, $(gaaa)_6$, $(cagc)_6$. Following hybridization, enrichment for microsatellites was performed by the addition of streptavidin magnetic beads (Dynabeads) and separation of the probe-microsatellite-containing-fragments was achieved with three non-stringent and four stringent wash steps. Eluted DNA was thereafter PCR amplified and cloned into bacterial vectors using the TOPO10 chemically competent cloning kit for sequencing (Invitrogen). Recombinant plasmids were plated on ampicilin-supplemented agar plates and allowed to grow overnight at 37ºC. We picked positive clones and infected 1ml of LB medium, which was subsequently grown for 16 hours at 37ºC. Thereafter 1 µl of 1/10 dilution of infected medium was used as template in a 25 µl colony PCR amplification. Polymerase chain reactions were carried out in 25 µl reactions consisting of 1 x PCR buffer, 2.5 mM $MgCl_2$, 2 mM of each DNTP, 5 µM of each primer (T3: 5' ATTAACCCTCACTAAAGGGA 3', T7: 5' TAATACGACTCACTATAGGG 3') and 0.3 U of Supertherm Taq polymerase (Southern Cross Biotechnology). Reaction cycles were performed on a Hybaid multiblock (Thermofast) and consisted of denaturation at 94ºC for 2 minutes, followed by 35 cycles of denaturation at 94ºC for 30 seconds, primer annealing at 55°C for 30 seconds, and elongations at 72ºC for 30 seconds, and

finally 72ºC for 7 minutes. PCR products were subsequently sequenced using only the T7 (forward) primer with ABI PRISM® BigDye™ chemistry according to the manufacturer's instructions. The nucleotide sequences of probed fragments were subsequently resolved on an ABI 3100 Capillary Sequencer (Applied Biosystems). Sequences were scanned for microsatellite repeats using the Staden software package (Bonfield *et al.* 2002), and primers were designed for clones containing microsatellite repeats using the Primer Designer software package (Sci Ed Software, http://www.scied.com/ses_pd5.htm). Potential microsatellite loci were scored for polymorphism, across ten individuals sampled widely from the distribution of the species, on polyacrylamide gels stained with GelStar (Cambrex Bio Science Rockland, Inc.), and labelled primers were ordered for polymorphic loci. Thereafter fragment size analysis of 130 individuals genotyped for the polymorphic loci was performed on an ABI 3100 Capillary Sequencer, using Genescan 3.1 and the LIZ-500 size standard (Applied Biosystems). Representative alleles from each locus were subsequently sequenced from homozygotes and have been deposited with Genbank (http://www.ncbi.nih.nlm.gov) under the accession numbers DQ020593-DQ020611. Calculation of levels of heterozygosity, and tests for deviation from Hardy-Weinberg equilibrium and the non-random association of alleles among loci (linkage disequilibrium) were performed in Arlequin v2.001 (Schneider *et al.* 2000).

A total of 323 positive colonies were picked and scanned for microsatellite repeats. Of the 323 positive colonies, 22 contained microsatellite repeats of sufficient length and for which primers could be designed. Polymorphism tests revealed that seven of these 22 were polymorphic. Seven pairs of labelled primers were subsequently ordered, yet one locus (Table 1, Gon 26.3) proved difficult to amplify consistently and was removed from further analyses. Of the six remaining loci two showed high levels of polymorphism (Gon120.3 & Gon65, Table 1), whereas the remaining four had intermediate levels of polymorphism. Sequencing of alleles indicated substitutions to be stepwise (Figure 1), yet Gon65 may exhibit a complicated mutation model evident in the homoplasy observed for allele size 224. Clearly such a complex repeat may have high degrees of homoplasy and might not be useful for inferences of population history. Levels of heterozygosity are low in four of the six loci as might be expected for a population that experiences large inter-annual population size fluctuations. Furthermore, there was no evidence for linkage disequilibrium (Exact test, 10000 permutations, mean P across loci-pairs = 0.456), yet significant deviations from Hardy-Weinberg equilibrium are evident in two of the six loci (Table 1). Clearly, demographic processes would have a profound effect on the observed levels of genotypic diversity and heterozygosity. Meglecz *et al.* (2004) have attributed the low levels of heterozygosity observed in many Lepidopteran microsatellite loci to the presence of null alleles. Since microsatellites in Lepidoptera have been shown to be inconsistent in amplification across individuals, the flanking

Table 1: Microsatellite loci isolated and optimized for *Gonometa postica*. Amplification conditions for each locus is provided, as are the levels of allelic diversity, observed and expected heterozygosity (H_O and H_E respectively), and statistical significance (P) of deviation from Hardy-Weinberg equilibrium. *conc.* = concentration. The Bonferroni corrected alpha rejection level is 0.008.

Locus	*Motif*	*Primer sequence (5'-3')*	*Size Range*	*Dye*	*Annealing Temp (°C)*	*$MgCl_2$ conc. (mM)*	*No. of alleles*	*H_O*	*H_E*	*P*
Gon6	$(caaa)_n caat(caaa)$	F : AGCCCATGTTACTCGTGAAG R: GGGTGGAAAGCCAGTTATCT	161-173	TM PET	59	2	4	0.78	0.69	0.02974
Gon60	$(ay)_6$	F: CTGAAGAATAGCCAGCTAGG R: TGTGAATCGTGCCAGCAATG	213-217	TM VIC	59	1.5	3	0.17	0.41	**0.00001**
Gon65	$(ac)_6(gc)_4(ac)_n(ag)_2(ac)_n at(ac)_n$	F: ACGTTCGTATAGGTTGACAT R: GAGCTAATTGGTGCATTCAT	192-242	TM PET	59	1.75	17	0.49	0.89	**0.00001**
Gon107	$(gtct)_n gtrt$	F: GAGCAACAGAAGCCAAAG R: CACCTTTCTTCTATGGCC	157-177	TM 6-FAM	59	1.5	5	0.50	0.51	0.59564
Gon26.3	$(gaaa)_{32}$	F: TTCGCTGTGGAGAACGGAAG R: CTCGTCTGTTGTGATGAG	274	TM 6-FAM						
Gon 120.3	$(tc)_3 tg(tc)_n$	F: CCCCTAACCTAACTGATG R: CTCGTCTGTTGTGATGAG	87-115	TM VIC	45	2	11	0.66	0.69	0.00218
Gon55.3	$(gtct)_n$	F: GCCTTCACACATGCAGTA R: GCCTTCACACATGCAGTA	100-116	TM PET	47	2	4	0.27	0.27	1.00

Gon65 $(ac)_6(gc)_4(ac)_n(ag)_2(ac)_nat(ac)_n$

224 (41.4) GCACACGCACACAAGT**ACACACACACACGCGCGCGCACACACACACACACACAC--AGAGACACACACACACACAC--ATACACACACACACAC**----------GTGCGCGCACGCGTGTGTGCACC : 37
224 (66.2) GCACACGCACACAAGT**ACACACACACACGCGCGCGCACACACACACACACACACACAGAGACACACACACACAC----ATACACACACACACAC**----------GTGCGCGCACGTGTGTGTGCACC : 37
226 (10.11) GCACACGCACACAAGT**ACACACACACACGCGCGCGCACACACACACACACACACACAGAGACACACACACACAC--ATACACACACACACAC**----------GTGCGCGCACGCGTGTGTGCACC : 38
238 (23.2) GCACACGCACACAAGT**ACACACACACACGCGCGCGCACACACACACACACACAC--AGAGACACACACACACACACACATACACACACACACACACACACAC**GTGCGCACACGCGTGTGTGCACC : 44

Gon60 $(ay)_6$

215 (10.1) CAGATGTACACTAACTCGTATATTGATAGCGTACTATAGCTCCGTACAATGGGTAGCTACGCTGGTTCTACCTCA**ACACACATAC--ATAT**CTTAGTTAGATAACTCAAATCACAGAAACCACATTGCTGGCACGATTCACA : 7
217 (40.3) CAGATGTACACTAACTCGTATATTGATAGCGTACTATAGCTCCGTACAATGGGTAGCTACGCTGGTTCTACCTCA**ACACACACACACACACAT**CTTAGTTAGATAACTCAAATCACAGAAACCACATTGCTGGCACGATTCACA : 8
217 (41.4) CAGATGTACACTAACTCGTATATTGATAGCGTACTATAGCTCCGTACAATGGGTAGCTACGCTGGTTCTACCTCA**ACACACACACACACACAT**CTTAGTTAGATAACTCAAATCACAGAAACCACATTGCTGGCACGATTCACA : 8

Gon107 $(gtct)_ngtrt$

161 (17.4) AATTTTTTGAATTTTT**GTCTGTCTGTCTGTCT----GTGT**GTTTGTTACATCATCACGTGAAAATTACTAGATGGATTGAATGCAATTTTCACTGATGTATTGCTAAAGGT : 5
161 (22.1) AATTTTTTGAATTTTT**GTCTGTCTGTCTGTCT----GTAT**GTTTGTTACATCATCACGTGAAAATTACTAGATGTATTGAATGCAATTTTCACTGATGTATTGCTAAAGGT : 5
165 (18.4) AATTTTTTGAATTTTT**GTCTGTCTGTCTGTCTGTCTGTAT**GTTTGTTACATCATCACGTGGAAATTACTAGATGTATTGAATGCAATTTTCACTGTTGTATTGCTAAAGGT : 6

Gon55 $(gtct)_n$

108 (10.1) GCCTTCACACATGCAGTAAAAGCTAAAAACGTGGTTTTAGAAACTTTT**GTCTGTCTGTCTGTCT**GTTGTACGCGATTCGCACTAAAACTATTGGGCGTGTTACTCAG : 4
104 (59.2) GCCTTCACACATGCAGTAAAAGCTAAAAACGTGGTTTTAGAAACTTTT**GTCTGTCTGTCT----**GTTGTACGCGATTCGCACTAAAACTATTGGGCGTGTTACTCAG : 3

Gon6 $(caaa)_ncaat(caaa)$

161 (22.1) GTTTTTGAGCCAAATCATTATATACCCA**CAAACAAA----------CAATCAAA**TTTTTCCTCTTTATAATATTAGTATAGACATAG : 4
161 (72.8) GTTTTTGAGCCAAATCATTATATACCCA**CAAACAAA----------CAATCAAA**TTTTTCCTCTTTATAATATTAGTATAGACATAG : 4
169 (10.4) GTTTTTGAGCCAAATCATTATATACCCA**CAAACAAACAAACAAA----CAATCAAA**TTTTTCCTCTTTATAATATTAGTATAGACATAG : 6
169 (11.1) GTTTTTGAGCCAAATCATTATATACCCA**CAAACAAACAAACAAA----CAATCAAA**TTTTTCCTCTTTATAATATTAGTATAGACATAG : 6
173 (18.2) GTTTTTGAGCCAAATCATTATATACCCA**CAAACAAACAAACAAACAAACAATCAAA**ATTTTCCTCTTTATAATATTAGTATAGACATAG : 7
173 (58.1) GTTTTTGAGCCAAATCATTATATACCCA**CAAACAAACAAACAAACAAACAATCAAA**ATTTTCCTCTTTATAATATTAGTATAGACATAG : 7

Gon120 $(tc)_3tg(tc)_n$

97 (117.1) CAGAAGT**TCTCTCTGTCTCTCTCTCTCTCTCTCTC**--TATTTTTCCTCCTCACAACAGACGAG : 14
97 (41.3) CAGAAGT**TCTCTCTGTCTCTCTCTCTCTCTCTC**--TATTTTTCCTCCTCACAACAGACGAG : 14
99 (40.2) CAGAAGT**TCTCTCTGTCTCTCTCTCTCTCTCTCTC**TATTTTTCCTCCTCACAACAGACGAG : 15

Figure 1: Sequenced alleles for each of the microsatellite loci isolated. For each sequence the following are provided: size on GeneScan, (sampled individual) DNA sequence and : number of repeats.

regions may be highly variable. We, however, believe the observed levels of heterozygosity in *G. postica* microsatellites are the result of extreme fluctuations in population size, since PCR amplification of the loci optimized in this study was consistent across individuals, and flanking sequences in the alleles we sequenced did not show unusual levels of variation as found by Meglecz *et al.* (2004). We are currently investigating the effects of a complex demographic history on microsatellite variability, and the ability to infer population processes, in southern African Wild Silk Moth populations.

Acknowledgements

This work was funded by the Mellon Foundation Grant to J. W. H. Ferguson, P. Bloomer and W. Delport, and by a National Research Foundation grant to P. Bloomer (Gun: 2053653). The opinions and views presented in this article are however, not necessarily those of the National Research Foundation. We would also like to thank Dr. J. Nagaraju, Centre for DNA Fingerprinting and Diagnostics, Hyderabad, India for hosting WD and affording him the opportunity to work with *Bombyx mori* microsatellite loci.

References

Bonfield J, Beal K, Jordan M, Cheng Y & Staden R (2002) The Staden Package. Medical Research Council. Laboratory of Molecular Biology. http://www.mrc-lmb.cam.ac.uk

Freddi G, Bianchi Svilokos A, Ishikawa H, Tsukada M (1993) Chemical composition and physical properties of *Gonometa rufobrunnea* silk. *Journal of Applied Polymer Science,* **48**, 99-106.

Meglecz E, Petenian F, Danchin E, Coeur D'Acier A, Rasplus J-Y, Faure E (2004) High similarity between flanking regions of different microsatellites detected within each of two species of Lepidoptera: Parnassius Apollo and Euphydryas aurinia. *Molecular Ecology*, **13**, 1693-1700.

Schneider S, Roessli D & Excoffier L (2000) ARLEQUIN version 2: A software for population genetic data analysis. Genetics and Biometry Laboratory, University of Geneva, Geneva, Switzerland.

Zane L, Bargelloni L, Patarnello T (2002) Strategies for microsatellite isolation: a review. *Molecular Ecology*, **11**, 1-16.

Chapter 4

The effect of large annual population size fluctuations on spatial genetic pattern in the continuously distributed African Wild Silk Moth (*Gonometa postica*)

"Genomes can be regarded as genetic archives that contain information about the past history of changes in demography and of natural selection."

Veuille & Slatkin 2002

The effect of large annual population size fluctuations on spatial genetic pattern in the continuously distributed African Wild Silk Moth (*Gonometa postica*)

Wayne Delport[1], Paulette Bloomer[1] & J. Willem H. Ferguson[2]

[1]Molecular Ecology and Evolution Programme, Department of Genetics, University of Pretoria, Pretoria, 0002, South Africa

[2]Centre for Environmental Studies, Department of Zoology and Entomology, University of Pretoria, Pretoria, 0002, South Africa

Abstract

Several insect species exhibit large inter-annual fluctuations in population size, yet the effect of such population size fluctuations on the inference of demographic parameters from population genetic data is not well understood. Although some theoretical models suggest that population size fluctuations result in an increase in spatial genetic structure as a result of local genetic drift, this is not consistent with observed spatial genetic patterns in many cyclical species. The African Wild Silk Moth (*Gonometa postica*) is a species that exhibits large inter-annual population size fluctuations in the Kalahari region of southern Africa. We used morphological estimates of dispersal ability, mtDNA sequencing, microsatellite genotyping and simulation modeling to determine the connectivity of populations in this species. Our results indicate high levels of gene flow as found in other cyclical species. Our simulation results confirm the occurrence of increased spatial genetic pattern at low dispersal distances, yet we cannot demonstrate the conditions under which gross overestimates of dispersal are obtained. Rather we observe that low levels of dispersal, less than 1% of the species total range, are sufficient to prevent the genetic drift effects of population size fluctuations in continuously distributed species. The implications of these results in interpreting spatial genetic patterns from cyclical species are discussed.

Keywords: population cycles, continuously distributed, isolation by distance, simulations

Introduction

Since its advent, the principle aim in the field of phylogeography has been the inference of population processes from spatially distributed genetic data (Avise *et al.* 1987). Such inferences are typically made from (i) reconstructed gene genealogies (Bermingham & Avise 1985, Avise *et al.* 1987), (ii) statistical tests of population structure and partitioning of genetic variance (Weir & Cockerham 1984, Excoffier *et al.* 1992, Raymond & Rousset 1995) or (iii) statistical modeling of hypotheses and inference of parameters (Kuhner *et al.* 1998, Beerli & Felsenstein 1999, 2001, Nielsen & Wakeley 2001, Knowles & Maddison 2002, Knowles 2004). The latter generally incorporates the observation that gene sorting is stochastic (Kingman 1982, Hudson 1990), and thus accounts for the substantial variability possible in the observed genetic data under identical demographic processes. These backward-through-time coalescent models provide a computationally tractable method for the estimation of population parameters from genetic data, given their assumptions. Assumptions typically include the dispersal model (island model: Wright 1931, lattice model: Malecot 1951), substitution model, neutrality of gene regions considered, and the underlying demographic model. The suitability of these analytical techniques for a species of interest is dependent on these assumptions. Several demographic models have been incorporated into coalescent methods, where one can estimate (i) gene flow from a structured population (Beerli & Felsenstein 1999, 2001), (ii) continuous population growth or decline in an unstructured or structured population (Kuhner *et al.* 1998, Kuhner *et al.* 2004) (iii) the divergence time of two populations that exchange/d migrants (Nielsen & Wakeley 2001) and (iv) immigration rates given a spatial range expansion (Ray *et al.* 2003, Exoffier 2004). Violation of the assumptions of these models, however, typically leads to discrepancies in the indirect estimation of gene flow from genetic data (Hastings & Harrison 1994, Leblois *et al.* 2003). In addition to these maximum-likelihood computational approaches, simulation approaches have been used to gain an understanding of the effect of temporal changes in dispersal and density on the inference of demographic parameters in continuous populations (Leblois *et al.* 2004). These computational and simulation approaches have, however, thus far only considered demographic processes that occur over extended genealogical periods. Several insect species that have short generation times exhibit large inter-annual population size fluctuations (Bowers *et al.* 1993, Ginzburg & Taneyhill 1994, Turchin 2003), determined by the interaction between exogenous and endogenous factors (Turchin 2003). In particular, the continuously distributed African Wild Silk Moth (*Gonometa postica*) exhibits large inter-annual population size fluctuations (Veldtman 2004, Delport 2005 Chapter 2). Yet the effect of such complex demographics on the population genetic inference of demographic parameters such as migration is uncertain.

G. postica is currently of economic interest in southern Africa since both this species and its sister species, *G. rufobrunnea*, have been shown to possess a silk fibre of exceptional quality (Freddi *et al*. 1993). To this end the initiation of an African Wild Silk industry has been proposed as a potential means of poverty alleviation in rural southern African communities. Attempts have already been made at initiating such an industry, yet a consistent complaint is the lack of a continuous local supply of cocoons, a problem that has already resulted in the demise of Shase Silk Pty (Ltd) in northwest Botswana. Since silk is processed after emergence of the adult moth, this problem is one of cyclical population dynamics and not harvesting impact. The life cycle of *G. postica* is characterized by two generations per annum, the first starting in September-October with the emergence of adult moths. Moths have no feeding mouthparts and only survive for three-five days (Hartland-Rowe 1992), during which breeding occurs. Eggs are laid, larvae emerge and pass through six instar larval stages in approximately five weeks, after which larvae pupate and enter diapause (Hartland-Rowe 1992). Diapause is interrupted in either February or September of the following year. Approximately 12-50% of the first generation emerge as adults in February and proceed with an additional population cycle. The impact of this variable second generation on the incidence of eruptions is, however, uncertain. In addition to the temporal cyclical nature of the species, abundance varies spatially within a season (Delport 2005 Chapter 2). It is clear that thorough population dynamics research is required to address eruptions in this species. Thus, a principle aim of the current research was to determine the dispersal ability *G. postica* using indirect morphological and molecular methods.

Preliminary analysis of population genetic data from *G. postica*, however, indicated that the inference of spatial genetic pattern might be problematic (Delport 2005 Chapter 3). This was due in part to the low levels of microsatellite allelic variation in the species (Delport 2005 Chapter 3), and the potential for a complex demographic history to disrupt the spatial genetic pattern. Complex demographic effects and their influence on demographic inferences from population genetic data in continuous populations are seldom evaluated (Leblois *et al.* 2004). Leblois *et al.* (2004) have shown that temporal and spatial fluctuations of demographic parameters (density, dispersal and population size) have little effect on the inference of neighbourhood size using Rousset's (2000) spatial regression method. Yet a temporal increase in density, even a relatively ancient flush of low magnitude, results in a significant overestimate of neighbourhood size (Leblois *et al.* 2004). In addition, the effect of demographic changes on the inference of population parameters is largely dependent on the timing of demographic events, where an event more than twenty generations in the past is considered to have little effect (Leblois *et al.* 2004). *G. postica*, however, exhibits annual

population size fluctuations of large magnitude, potentially two to three orders (Veldtman 2004). Neighbourhood size has been proposed as a suitable statistic for inferring levels of gene flow in continuously distributed species (Rousset 2000), yet the effect of annual population size fluctuations on its inference is unknown. Theoretically, the degree of dispersal is likely to determine the effects of population size fluctuation on estimates of neighbourhood size. When dispersal is low, population size fluctuations could generate population genetic structure between demes, the result of local genetic drift (Wright 1940). However, when dispersal is high, size fluctuations could have a homogenizing effect on spatial genetic pattern as a result of global genetic drift and high connectivity between populations. Inferring the position of a species along this continuum using genetic data is a question that has not yet been explored, the result of which would be instrumental in our understanding of the effects of complex demographic processes. Therefore, in this manuscript we evaluate the dispersal ability of *G. postica* using indirect morphological and molecular methods and interpret these results in light of the effects of population size fluctuations on spatial genetic pattern, as derived with simulation modeling. Our results show a very weak signal of spatial genetic structure in *G. postica*. Furthermore, simulation results suggest that low levels of dispersal in continuously distributed species are sufficient to prevent an increase in population genetic structure as a result of population size fluctuations. These results are discussed in relation to observed spatial genetic patterns detected in other cyclical species.

Materials and methods

Study site and morphological data analysis

G. postica is an eruptive species that occurs in the Kalahari region of southern Africa (Figure 1). Although the species has been recorded in other areas, east and south of the distribution depicted, the Northern Cape province of South Africa and Kalahari region of southern Africa are considered the core region (Veldtman *et al.* 2002), where the species is regularly recorded in large numbers. We collected pupae (cocoons) from trees during the latent phase of their development in June 2002. Adult moths were then allowed to emerge in the laboratory and were frozen at -20°C for future genetic analyses. We estimated an indirect measure of dispersal-ability based on morphology. Wing load (i.e. the ratio of body mass to total wing area) has previously been used as a proxy for estimating dispersal or flying ability of Lepidoptera (Casey & Joos 1983). We pinned and photographed one fore- and one hindwing from each of 153 individuals (57 males, 96 females) at 2 mega pixel resolution with a FujiFilm FinePix S304 digital camera at constant height. Eight calibration photographs were taken at random intervals during the course of digitizing wings. Total wet body mass was measured, using a Scaltec Sac51 digital scale, before removal of wings. Dry

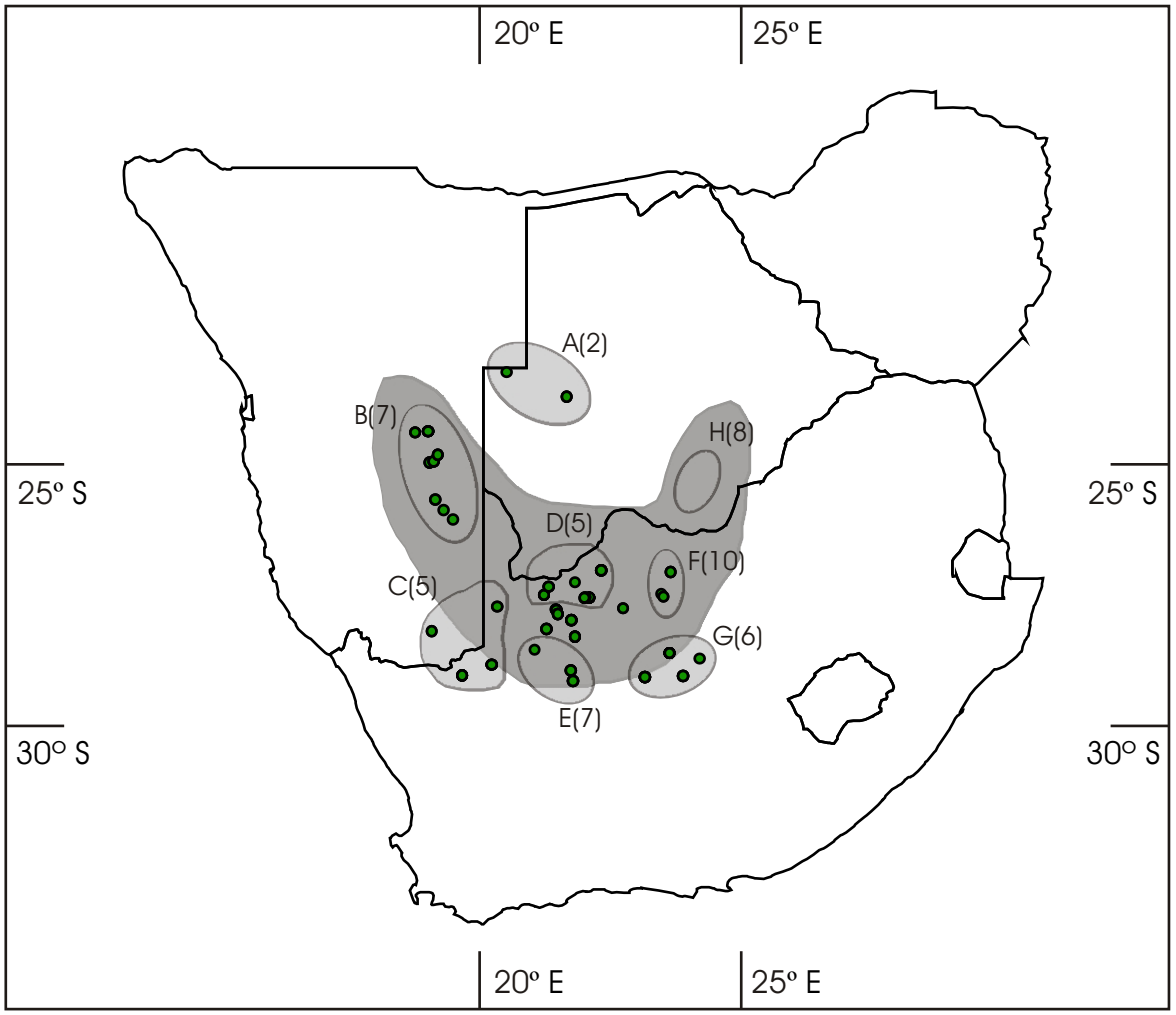


Figure 1: Distribution of *Gonometa postica* in southern Africa and sampling localities for mtDNA and microsatellite samples. The distribution of eruptive populations is depicted as a large gray polygon, whereas small light gray polygons depict locations of sampling sites for mtDNA analysis (with sample sizes in parentheses). Microsatellite sampling localities are represented by small filled circles.

mass was not determined since the moths were frozen at -20ºC for subsequent DNA extraction and analyses. The sexes of the moths were easily scored due to the large sexual size dimorphism evident in the species (Veldtman *et al.* 2002). Total wing area was measured from digitized images. Images (JPEG format) were first converted to grayscale images in ImageJ (Rasband 1997) and wing areas were measured with the particle analyze tool (where all particles greater than 10000 pixels were analyzed). Scale (in mm) was set from calibration images.

Genetic sampling and laboratory methods

Total genomic DNA was extracted from 137 individuals collected from 23 localities (Figure 1, mean 4.9 individuals per locality) using a Qiagen Dneasy extraction kit. Partial fragments of the mitochondrial DNA control region were PCR-amplified, for 50 individuals collected across the distribution range (Figure 1), in a total volume of 50µl in thin-walled microcentrifuge tubes. The reaction mix contained approximately 50ng genomic DNA, 2mM $MgCl_2$, 1x reaction buffer, 0.2mM of each of four nucleotides, 1.5U Super-therm DNA polymerase (Southern Cross Biotechnology) and 12.5 picamol of each primer (12S-332+ Taylor *et al.* 1993: 5' TAGGGTATCTAATCCTAGTT 3'; *Bombyx mori* position 9582, tRNA-ile-r: 5' ATCAGAATAATCCTTTATTCAGGC 3'; *B. mori* position 10475). A Geneamp PCR system 9700 (Applied Biosystems) was used to cycle the reaction mix through the conditions: 94°C for 2min; 35 cycles of 94°C for 30 seconds, 59°C for 30 seconds and 72°C for 45 seconds; followed by an extended elongation at 72°C for 10 minutes. PCR products were ethanol precipitated and cycle sequenced using BigDye (Applied Biosystems) and electrophoresed on an ABI3100 capillary sequencer (Applied Biosystems), yielding a 250bp fragment of the control region. Difficulties with sequencing through the AT-rich repeat region, characteristic of the insect mtDNA control region, prevented the sequencing of the full 487bp PCR product. Mitochondrial DNA sequences are available in Genbank (Accession numbers: DQ020593-DQ020611). In addition to DNA sequencing we genotyped 137 individuals for six polymorphic microsatellite loci (Gon6, Gon60, Gon65, Gon55, Gon107, Gon120). Details of protocols for microsatellite development and amplification conditions are detailed in Delport (2005 Chapter 3).

Statistical genetics methods

Before conducting advanced statistical inference we first tested for neutrality of mutations, in the mtDNA data, using Tajima's (1989) D and Fu & Li's (1993) F_s test statistics. These statistics are based on the premise that (i) in a gene under selection, θ estimated from segregating sites will be substantially greater than θ estimated from nucleotide diversity, since rare mutations selected

against are down-weighted in the calculation of the latter (Tajima's *D*), or (ii) purifying selection is evident as an excess of mutations at the tips of a genealogy, versus those occurring on internal branches of the genealogy (Fu & Li's F_s). Failure to reject the null hypothesis of neutrality is evidence for the absence of selection in the gene considered; yet the rejection of neutrality could be the result of either selection or the demographic process of rapid population growth following a bottleneck (Tajima 1989). Mitochondrial DNA sequence data were used to draw an allele network, a hypothesis of the genealogical relationships among alleles, using TCS (Clement *et al.* 2000). Before making inferences from the allele network, we used a process of eliminating homoplasy, where older alleles of higher frequency have greater probability of being interior nodes (Posada & Crandall 2001), and where similar alleles are more likely to be derived from adjacent or closely situated localities. A structured population is not consistent with the isolation by distance model most appropriate for this species, yet due to insufficient mtDNA sample sizes we grouped localities based on geographic proximity and the temporal persistence of eruptions (Delport 2005 Chapter 2, Figure 1). *F*-statistics, based on pairwise differences, were calculated using Arlequin (Schneider *et al.* 2000). Finally, we used the mtDNA data to test for the existence of population structure with 1000 permutations of the chi-square test (Roff & Bentzen 1989). Microsatellite data were evaluated for linkage disequilibrium, the test of non-random association of alleles at different loci, and for Hardy-Weinberg equilibrium using the Arlequin software package (Schneider *et al.* 2000).

The statistical methods we used to further analyze the microsatellite data were based on a continuous population and isolation by distance model, where the probability of identity in state between two neutral genes decreases with the distance between them (Wright 1943, 1969). This model is most suitable when analyzing data from species that are continuously distributed, or where demes or subpopulations cannot be easily demarcated. The analyses were subdivided into three approaches: (i) spatial autocorrelation statistics, (ii) spatial regression statistics, and (iii) hierarchical *F*-statistics. Although, the calculation of *F*-statistics is not based on the isolation by distance model *per se*, the method of the sequential pooling of individuals into populations allows the inference of the scale at which populations may be structured.

*(i) Spatial autocorrelation statistics: S*patial autocorrelation methods were used to assess the spatial distribution of allelic frequencies among localities. Spatial autocorrelation methods typically involve the calculation of some spatial statistic, such as Moran's *I* (Sokal & Oden 1978, Epperson & Li 1996) at subsequent distance intervals. Theoretically, a decrease in the statistic with distance is indicative of the degree of isolation by distance (Hardy & Vekemans 1999, Diniz-Filho & Telles 2002,Vekemans & Hardy 2004). Since the choice of distance intervals can bias the interpretation of

results in the spatial analyses (Fenster *et al.* 2003), Hardy & Vekemans (2003) have suggested the use of two statistics that prevent this potential bias. The % partic is the proportion of all individuals represented at least once in the particular distance interval; whereas the CV partic is the coefficient of variation of the number of times each individual is represented in the distance interval (Hardy & Vekemans 2003). As a rule of thumb the % partic should be greater than 50% and CV partic less than or equal to 1 for each distance interval. We tried alternate distance intervals and chose the set which satisfied these conditions (Table 1). Spatial autocorrelation statistics were calculated for each of the above distance classes using SPAGeDi (Hardy & Vekemans 2003) and the relationship between spatial autocorrelation coefficients and distance (correlograms) was plotted to quantify isolation by distance.

(ii) Spatial regression statistics: The spatial regression of genetic distance with geographic distance was used to estimate neighborhood size (Nb). Rousset's (1997) multilocus estimator of an $F_{st}/(1-F_{st})$ between populations, was plotted against the log of geographical distance, and regression statistics calculated. We used a population-based statistic over an individual-based statistic (Rousset 2000) since we had sampled multiple individuals from single trees. An individual-based approach (Rousset 2000) would require the exclusion of data where more than one individual was sampled at a particular tree, and thus we treated individual trees as populations, and included populations located 1 to 500 km apart with more than 3 individuals. We calculated multi-locus estimates of pairwise differentiation, regressed against the logarithm of geographic distance and tested the significance of the observed correlation, between genetic and geographic distances, with a randomized permutation procedure (10000 permutations). *P*-values were estimated as the proportion of the permuted regression values that were greater, and thus indicative of greater isolation by distance, than that observed. Neighborhood size was calculated as the inverse of the slope of the regression between genetic and the logarithm of geographic distance. All spatial regression analyses were performed with GENEPOP v3.4 (Raymond & Rousset 1995).

(iii) Hierarchical F-statistics: Another potential method for detecting the scale at which a population may be structured is that proposed by Goudet *et al.* (1994), known as hierarchical F-statistics. The method comprises the sequential pooling of individuals into populations by increasing the extent of each population in each subsequent level, until all individuals are included as a single population. At each level *F*-statistics are calculated and plotted against the level of pooling. Although the method used to pool individuals is subjective, pooling on the basis of geographic distance is acceptable (Goudet *et al.* 1994, Arnaud *et al.* 2001). We used six levels for the calculation of *F*-statistics: within 1km, 5km, 10km, 50km, 100km and finally northern versus

Table 1: Statistics for the optimal distance classes chosen for spatial correlogram analysis. "Maximum distance" is the upper bound of each distance class, "Number of pairs" is the number of pairwise comparisons within the distance class. "% partic" and "CV partic" are indications of the proportion of all individuals utilised to calculate the statistics of interest within each distance class.

Distance Class	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Maximum Distance (km)	10	50	100	150	200	250	300	400	500	600
Number of pairs	384	1104	1159	1608	1473	556	753	1453	664	26
% partic	95.6	94.1	91.9	93.4	100	90.4	96.3	100	64.7	11.0
CV partic	0.72	0.71	0.82	0.65	0.60	1.14	0.90	0.86	1.24	3.64
Mean distance (km)	1.5	38.8	76.1	128	171	225	278	344	448	511
Mean ln(distance)	-0.35	3.62	4.31	4.85	5.14	5.41	5.62	5.84	6.10	6.24

southern populations (localities A-B and C-G in Figure 1 respectively). For each pooling level, adjacent populations within the specified distance of one another were combined as a single population. Weir & Cockerham's (1984) *F*-statistics, F_{ST} and F_{IS}, were calculated, using custom software for each sequential grouping, and 95% confidence intervals were obtained by bootstrapping over loci.

Simulation model

To investigate the relationship of dispersal distance and the inference of spatial genetic pattern under fluctuating population size, we constructed a custom forward-through-time simulation lattice model of the process of population turnover in *G. postica*. The lattice model (Malécot 1951), in which each node represents an individual, has been used extensively for both analytical (Rousset 2000) and simulation (Sokal & Wartenberg 1983, Epperson 1995, Leblois *et al.* 2003, Leblois *et al.* 2004) approaches to modeling continuous populations. We used a 500 x 500 lattice model to simulate the effect of annual population size fluctuations on the inference of Rousset's (2000) pairwise-individual estimate of neighbourhood size. The model comprised four steps; (i) population size was determined stochastically, (ii) breeding pairs were formed between individuals and progeny produced, (iii) progeny genotypes were chosen at random from their parents, and mutated, (iv) lattice positions for progeny were chosen from empty lattice nodes in the subsequent generation.

(i) Population size in the subsequent generation was determined with a discrete-time Ricker model (Ricker 1954) of population growth. This discrete analogue of the logistic growth model was used since the model is capable of both stability, and of limit cycles and chaos (Turchin 2003). The Ricker model is defined by:

$$N_{t+1} = N_t e^{r_t(1-\frac{N_t}{k})}$$

where, N_t is the population size at time t, r_t the population growth rate, and k is the carrying capacity. Stochasticity, and thus population size fluctuations, was introduced into the model by drawing a parameter ε_t, from a normal distribution with mean = 0, and variance σ^2, at each generation and calculating a new growth rate (r_{t+1}) by adding ε_t to r_t (Turchin 2003). Through changing the variance, σ^2, the degree of population size fluctuations could be altered. We simulated population turnover under three levels of variance (constant size, $\sigma^2 = 0.1$, $\sigma^2 = 0.001$) to observe the effects of size fluctuations on the inference of population statistics.

(ii) Once population size had been determined, breeding pairs were chosen by first choosing a

female, and then choosing a male located within *d* lattice units of the female. A single progeny was produced from the union of male and female.

(iii) The genotypes for each progeny were constructed by randomly choosing an allele each from the female and male of the breeding pair. Mutations were implemented as a stepwise mutation model (SMM: Ohta and Kimura 1973) with locus specific mutation rates drawn randomly from a gamma (2, 2.5 x 10^{-4}) distribution. This procedure accounts for the observation that mutation rates at individual loci are variable, yet with a mean mutation rate considered to be the average in several taxa (Leblois *et al.* 2003). More advanced mutation models have been proposed (Generalised Stepwise Mutation Model, e.g. Pritchard *et al.* 1999, Infinite Allele Model: Kimura & Crow 1964, K-Allele Model: Crow & Kimura 1970), yet Leblois *et al.* (2003) have shown that the impact of more advanced mutation models in the calculation of neighbourhood size is negligible. Thus we have only implemented a SMM in this model.

(iv) Finally, the lattice position of the progeny was chosen randomly from within *d* lattice units of the breeding female. If the progeny was unable to find a lattice position within a specified search time, it was abandoned and not carried through to the subsequent generation. The above procedure of finding a breeding pair, creating genotypes and populating the subsequent generation was repeated until the calculated population size had been reached. By sampling breeding pairs with replacement, and limiting the search time (10000 permutations) for progeny to fill a lattice node, variance in reproductive success of individuals was introduced.

Population turnover was simulated over 1000 generations for three levels of dispersal (d = 2, 4, 6), where *d* is the dispersal parameter and represents the maximum number of lattice units for breeding pair formation and natal dispersal. Although the increase in *d* is small, it translates into large increases in the breeding pair and progeny lattice search area (4 x 4, 8 x 8, 12 x 12). Preliminary runs of the simulation model under constant population size indicated that these three dispersal parameters generated neighbourhood sizes of approximately 10, 25 and 50, respectively. Thus we started the simulation from a random distribution of alleles among nodes, and simulated restricted dispersal (d = 2, 4, 6) for 1000 generations. At each generation pairs were formed, genotypes constructed and progeny positions were chosen, as described above. Six microsatellite loci were simulated, where the starting number of alleles in each locus was drawn from a gamma (6, 5) distribution. This method accounted for the observed patterns of allelic diversity in *G. postica*, where a few loci were highly diverse, yet most had moderate to low diversity (Delport 2005 Chapter 3). The three starting populations were subsequently used in all simulations of fluctuating population size. When testing for the effect of population size changes on a calculated statistic such as neighborhood size, two potential sources of variation are encountered: variation in the

calculation of the statistic from a sub-sample of the population, and variation in the statistic as a result of demographic instability. Since we are interested in inferring the latter, we used a permutation procedure (n = 1000) where neighbourhood size was calculated from the generated starting population, with 200 samples drawn randomly at each permutation. In this way variance in the calculation of neighborhood size under a constant population size could be determined (Table 2), and could thus be accounted for when observing trends under fluctuating population size. We simulated population turnover for 100 generations under three levels of dispersal, three neighbourhood size starting conditions and three levels of population size fluctuation.

Results

Morphological results

Consistent with the size dimorphism results of Veldtman *et al.* (2002) we detected significant differences between male and female body mass, hind wing area, forewing area and total wing area (Table 3). Females are approximately six times heavier than males, yet have approximately only four times more wing surface area than males (Table 3). This result translates into a statistically significant higher wing load for females, than for males.

General statistics

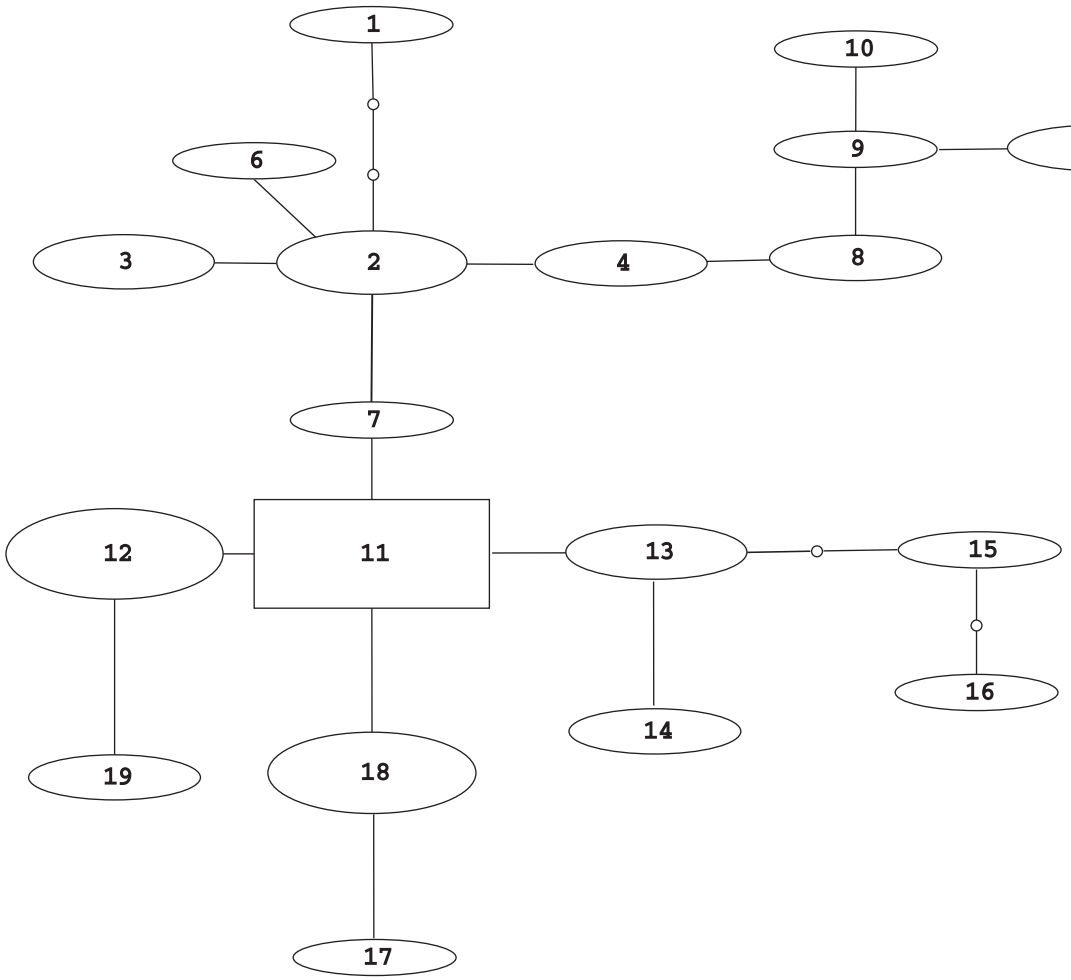
Statistical tests for deviations from neutrality indicated that the AT rich mtDNA region sequenced did not show significant deviations from neutrality (Tajima's D = -0.875, P = 0.19; Fu's F_s = -2.821, P = 0.067) where significance was determined with 1000 permutations. Both neutrality statistics, however, were negative, which is indicative of an excess of homozygotes. Given that *G. postica* populations experience large annual population size fluctuations this result is expected. However, whether the observed deviations from neutrality, although non-significant, are the result of demographic instability or a selective sweep would require analysis of another unlinked marker. The allele network (Figure 2) indicates little genetic structure among localities, with a few high frequency widely distributed alleles. Allele 11 in particular is found in six of the eight locality groups (Figure 2). Again this pattern is typical of a population that has experienced population growth (Rogers & Harpending 1992, Avise 2000). Although the overall pattern appears to be panmixia, some populations have unique alleles. *F*-statistics (Table 4) indicate that most locality groups have high levels of maternal gene flow, yet groups G and H exhibit significant structure when compared with the remaining locality groups. Finally, the probability of obtaining the observed distribution of alleles among localities at random is extremely low (X^2 = 161.63, *P* <

Table 2: Sampling variance in the calculation of neighbourhood size (Nb) from starting populations used in the simulation model. Mean and standard deviations of 1000 permutations of the calculation of neighbourhood size (Nb) using 200 samples drawn at random from each of the three starting populations (Nb = 10, Nb = 25, Nb = 50).

	Nb = 10	**Nb = 25**	**Nb = 50**
Mean	13.08	23.11	39.14
Sd	2.39	6.27	17.62

Table 3: Indirect morphological estimates of dispersal ability in *Gonometa postica*. Mass, hindwing, forewing and total wing areas are shown, as is the wing load; the ratio of body mass to total wing area. The results of a t-test, for unequal sample sizes, of the difference in means between male and female morphological characteristics is shown, $F_{(df1,df2)}$, where *df* = degrees of freedom, as is the associated *P* value.

	male		**female**		$F_{(1,149)}$	***P***
	mean	**sd**	**mean**	**sd**		
Mass (g)	0.533	0.111	2.973	0.652	769.587	<<0.001
Hindwing area (mm^2)	63.013	12.778	268.366	52.585	823.228	<<0.001
Forewing area (mm^2)	124.936	20.804	446.481	85.430	764.604	<<0.001
Total wingarea (mm^2)	187.949	32.758	714.847	136.210	808.305	<<0.001
Wing load (g/mm^2)	0.00291	0.001	0.00420	0.001	85.595	<<0.001
Wing load (N/m^2)	28.64	7.63	41.24	8.34		



Allele	Locality (frequency)	Total Frequency
1	A	1
2	A (1), E (2), F (1)	4
3	B (1), F (2)	3
4	H (2)	2
5	C (1), D (1)	2
6	B (1)	1
7	B (1)	1
8	B (1), F (1)	2
9	C (1)	1
10	B (1)	1
11	B (2), C (1), D (1), E (2), F (2), G (1)	9
12	F (1), G (5), H (1)	7
13	F (1), H (2)	3
14	C (2)	2
15	H (1)	1
16	H (1)	1
17	D (1)	1
18	E (3), F (2), H (1)	6
19	D (2)	2

Figure 2: Mitochondrial DNA allele network constructed in TCS (Clement *et al.* 2000). Frequencies of each allele are depicted by size and in the accompanying table. Frequencies of each allele at each mtDNA sampling locality group are presented.

Table 4: Calculation of mtDNA pairwise-population F_{ST} values using Arlequin (Schneider *et al.* 2000) for subpopulations/demes defined in Figure 1. Subpopulations A and B were combined for the purpose of this analysis. F_{ST} values are presented below the diagonal, and significance of deviations of F_{ST} from zero are shown above the diagonal (10000 permutations), where significance at the 0.05 level is indicated in bold. Sample sizes for each subpopulation/deme are indicated in parentheses.

	A&B (9)	**C (5)**	**D (5)**	**E (7)**	**F (10)**	**G (6)**	**H (8)**
A&B		0.46273	0.46203	0.24126	0.93110	**0.00129**	**0.00040**
C	0.01656		0.61439	0.10583	0.13108	**0.01495**	**0.02158**
D	0.01656	0.02174		0.10751	0.12821	**0.01346**	**0.02099**
E	0.03724	0.12454	0.12454		0.54965	**0.00505**	**<0.0001**
F	-0.04553	0.04306	0.04306	-0.02728		**0.01257***	0.14979
G	**0.28605**	**0.38120**	**0.38120**	**0.41312**	**0.24121**		**0.00416**
H	**0.06014**	**0.09744**	**0.09744**	**0.16667**	0.03903	**0.28292**	

0.001), as determined with permutations of the chi-square test (Roff & Bentzen 1989). Distributions of microsatellite allele frequencies (Figure 3) indicate that most loci have high frequencies of one to two common alleles, yet low frequencies of the remaining alleles. Of the six microsatellite loci, four (Gon6, Gon60, Gon65, Gon120) showed significant deviations from Hardy-Weinberg equilibrium (Table 5), of which two (Gon60, Gon65) were highly significant ($P < 0.001$). No evidence for linkage among loci was detected.

Statistical genetics methods

(i) Spatial autocorrelation

Spatial autocorrelation statistics plotted against distance showed no clear pattern of isolation by distance (Figure 4), with most correlograms showing random fluctuations in spatial autocorrelation statistics with distance. Furthermore, results from permutation tests of the regression of spatial statistics against ln (distance) indicate that combined-locus correlograms do not indicate a significant association of spatial statistics with distance (Table 6). However, some locus-specific correlograms, specifically Gon55, do show significant regressions with distance for the two kinship statistics (Loiselle *et al.* 1995, Ritland 1996) and Moran's I (Table 6). Gon55, a perfect tetranucleotide repeat, exhibits a spatial pattern for some statistics that is significantly greater than that of a permuted sample. Yet, examination of the spatial regression pattern on correlograms (Figure 4) indicates that these statistics show a substantial increase or decrease at the 500 or 600km level. The % partic and CV partic (Table 1) at these distance levels indicate that fewer individuals are represented at these distance classes (low % partic), and some individuals may be overrepresented (high CV partic). This is most likely the cause of the observed deviations from no pattern of spatial autocorrelation that is predominant in the data.

(ii) Spatial regression

The regression analysis of pairwise $F_{ST}/(1-F_{ST})$ against geographic distance yielded a positive relationship (Figure 5). However, permutation tests determined that the probability of a random correlation between geographic and genetic distances being greater than that observed, was intermediate ($P = 0.432$) and thus the observed correlation is non-significant. Calculation of neighbourhood size from the slope of the regression yielded an estimate of 167 individuals. Given a demographic density estimate, and the calculated neighbourhood size, it is possible to infer dispersal (Rousset 2000). McGeoch *et al.* (2004) provide preliminary density estimates of 150-1075 cocoons per hectare (0.015 – 0.1075 cocoons/m^2). However, since *G. postica* cocoons are parasitized extensively (Veldtman *et al.* 2004) we adjusted the density for no loss, 25% loss and

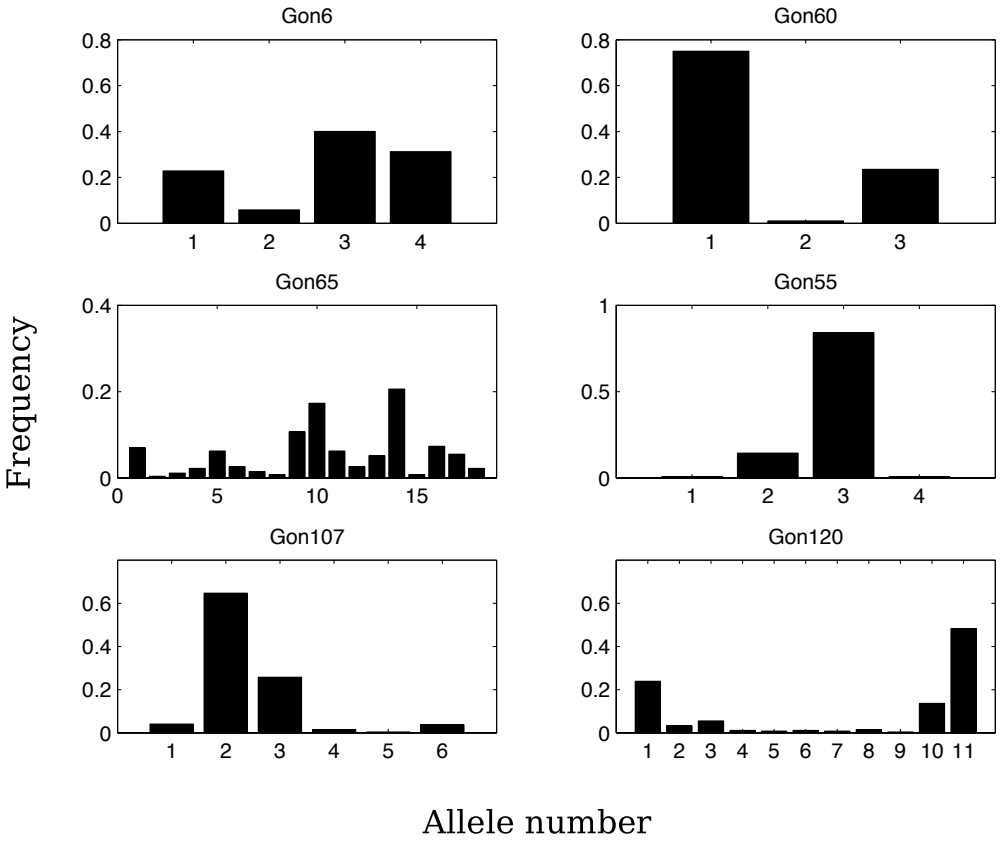


Figure 3: Distribution of microsatellite allele frequencies for each of the six loci analysed for 137 *G. postica* samples collected across the distribution of the species.

Table 5: Tests of the deviation from Hardy-Weinberg, the random-association of alleles within diploid individuals as calculated using Arlequin v2.0 (Schneider *et al.* 2000). Significant deviations from Hardy-Weinberg equilibrium for each microsatellite locus was calculated with 100000 steps in the Markov chain, with 1000 dememorisation steps. H_0 = observed heterozygosity, H_e = expected heterozygosity, n = number of alleles per locus.

	Ho	***He***	***P***	***n***
Gon6	0.780	0.690	0.029*	4
Gon60	0.162	0.391	< 0.001**	3
Gon65	0.493	0.893	< 0.001**	17
Gon55	0.272	0.277	1.000	5
Gon107	0.500	0.514	0.606	11
Gon120	0.662	0.694	0.002*	4

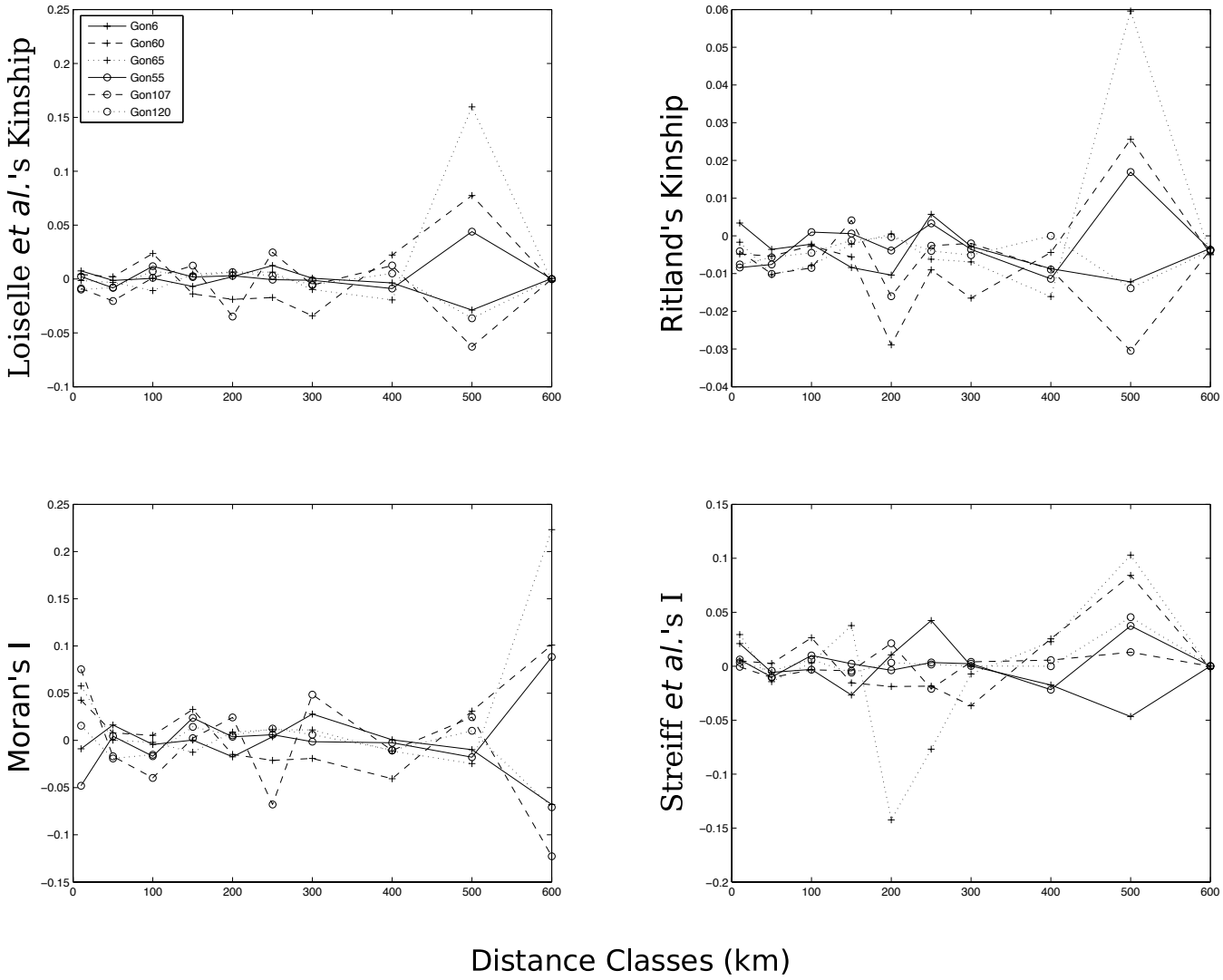


Figure 4: Correlograms of spatial autocorrelation statistics plotted against geographic distance classes. In each case the spatial autocorrelation statistic calculated is plotted against increasing distance for each of the six loci.

Table 6: Results of permutation tests for the regression of spatial autocorrelation statistics against ln (distance). K_L= Loiselle *et al.*'s (1995) kinship; K_R = Ritland's (1996) kinship; I_k = Moran's I (Sokal & Oden 1978); I = Streiff *et al.*'s (1998) I. For each case the probability, *P*, that a permuted regression statistic is greater than that observed (obs < per), less than that observed (obs > per) and not equal to that observed (obs <> per) is represented. * = significance at 0.05; ** = significance at 0.01. obs = observed regression statistics, per = permuted regression statistic. A Bonferroni corrected alpha level of 0.0125 was used for significance at the 0.05 level.

	K_L	**K_R**	**I_k**	**I**
All Loci				
P (H: obs < per)	0.57	0.3	0.58	0.35
P (H: obs > per)	0.43	0.7	0.42	0.65
P (H: obs <> per)	0.85	0.59	0.84	0.71
Gon6				
P (H: obs < per)	0.92	0.84	0.93	0.47
P (H: obs > per)	0.08	0.16	0.07	0.53
P (H: obs <> per)	0.16	0.33	0.15	0.94
Gon60				
P (H: obs < per)	0.2	0.25	0.21	0.21
P (H: obs > per)	0.8	0.75	0.79	0.79
P (H: obs <> per)	0.41	0.51	0.42	0.41
Gon65				
P (H: obs < per)	0.04	0.04	0.04	0.18
P (H: obs > per)	0.96	0.96	0.96	0.82
P (H: obs <> per)	0.09	0.08	0.09	0.37
Gon55				
P (H: obs < per)	0.99	0.99	0.99	0.87
P (H: obs > per)	**<0.01****	**<0.01****	**0.01***	0.13
P (H: obs <> per)	0.03	**<0.01****	0.03	0.25
Gon107				
P (H: obs < per)	0.91	0.7	0.92	0.64
P (H: obs > per)	0.09	0.3	0.08	0.36
P (H: obs <> per)	0.18	0.59	0.17	0.71
Gon120				
P (H: obs < per)	0.66	0.74	0.66	0.62
P (H: obs > per)	0.34	0.26	0.34	0.38
P (H: obs <> per)	0.69	0.51	0.68	0.75

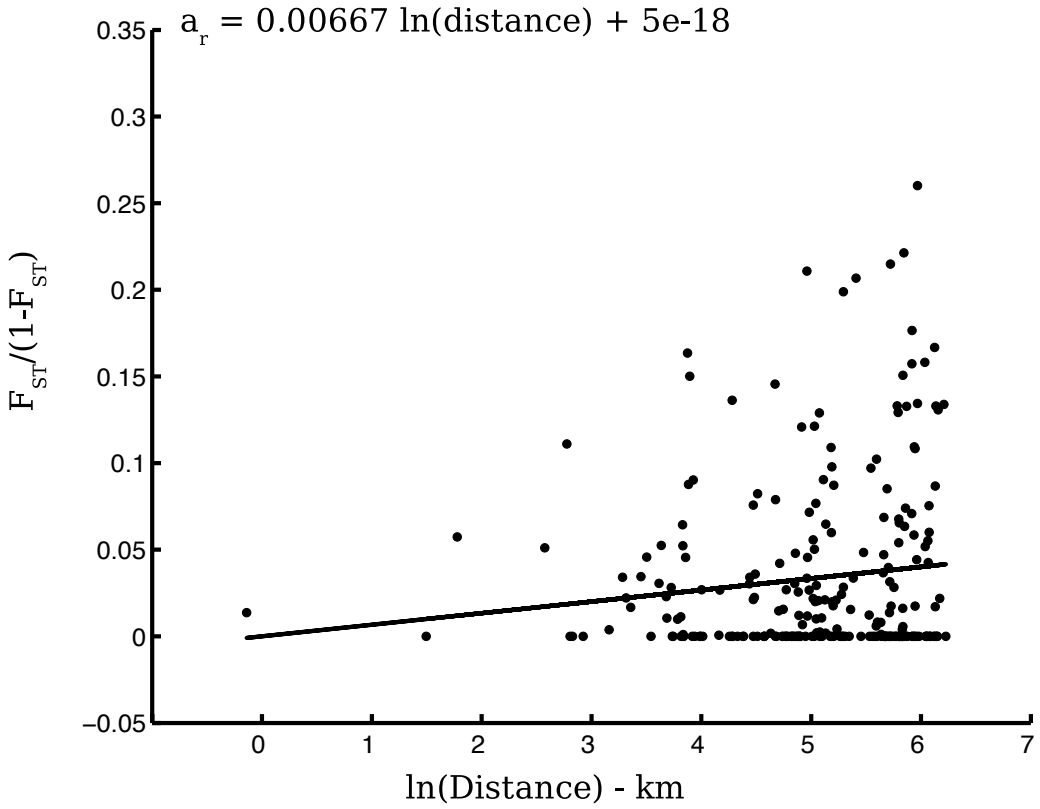


Figure 5: Genetic differentiation in the African Wild Silk Moth, *Gonometa postica*, represented as the regression of pairwise genetic distance against the natural logarithm of distance (km). a_r = 0.00667 ln(distance) + 5e-18 . Permutations (10000) indicated that the probablity of obtaining a randomised correlation greater than that observed was moderate (P = 0.433).

50% loss to parasitism and predation. These corrected densities of adults/m^2 have not been sufficiently replicated for certainty; yet do provide a potential dispersal range of *G. postica*, given the observed microsatellite data. The adjusted densities, combined with a neighbourhood size of 167 individuals, suggest a mean parent-offspring dispersal distance of between 11.1 and 42.1m.

(iii) Hierarchical F-statistics

Hierachical *F*-statistics indicated a slight increase in F_{IS} with the subsequent pooling levels (Figure 6a). Since F_{IS} is a measure of deviation from Hardy-Weinberg within subpopulations, an increase in F_{IS} is expected over subsequent pooling levels. This result is the consequence of combining non-interbreeding subpopulations, and thus violating the Hardy-Weinberg equilibrium assumption of random mating within populations (Wahlund effect, Wahlund 1928). In contrast, subsequent pooling levels show a decrease in F_{ST} (Figure 6b), especially evident after the 4^{th} level, where localities occurring within 50km of one another were pooled. Since F_{ST} is a measure of genetic differentiation over subpopulations, the rapid decrease in F_{ST} at this point is indicative of where the pooling level represents an essentially panmictic population, i.e. between subpopulation genetic differentiation is pooled into a single panmictic unit. Furthermore, we tested whether the mean permuted F_{ST} values at pooling levels 4 and 5 differed significantly using a one-way-ANOVA. The mean of permuted F_{ST} values at pooling levels 4 and 5 were significantly different, and thus reinforce the slight decrease in overall population genetic structure observed at this pooling level. Thus, although the overall levels of F_{ST} are low, the pattern observed using hierarchical F_{ST} analysis does allow the inference of the geographical extent of a subpopulation.

Simulation model

It is evident that quite different patterns of population size fluctuation can be achieved through adjusting the parameter σ^2 in the calculation of per generation population size (Figure 7). These population size fluctuations translate into different patterns of change in neighbourhood size over time dependent on the dispersal parameter. First, when population size is kept constant, low dispersal parameters (d = 2, d = 4) appear to reach equilibrium at neighbourhood sizes of approximately 10 and 25-30 respectively (Figure 8, Table 7), irrespective of the neighbourhood size of the starting population. A larger dispersal parameter (d = 6) shows a steady increase in neighbourhood size over time, and when run over increased number of generations (1000) achieves equilibrium at a neighbourhood size of 40-50 individuals (Table 7). When subjected to population size fluctuations neighbourhood sizes show (i) a steady decrease, (ii) a rapid decrease or (iii) no apparent difference in comparison to constant population size. Steady decreases in neighbourhood size are observed at a small starting neighbourhood size (Nb = 10) and low dispersal (d = 2), for

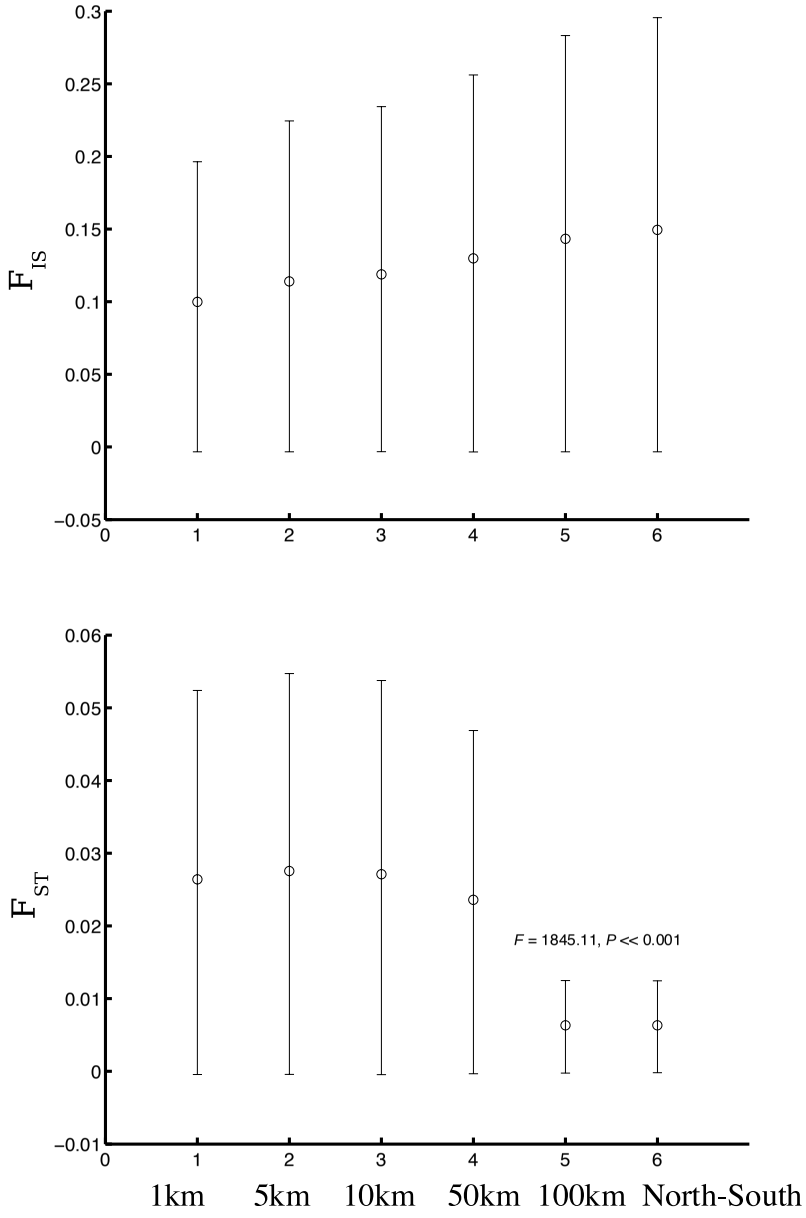


Figure 6: Changes in F_{ST} and F_{IS} calculated over six polymorphic microsatellite loci for each subsequent pooling level in the hierarchical F_{ST} analysis. Pooling levels are described in the methods. Mean and standard errors of the mean, as determined from bootstrapping over loci, are shown. Results of a one-way ANOVA of bootstrapped F_{ST}'s for pooling levels 4 and 5 are indicated by the test-statistic, *F*, and associated *P* value.

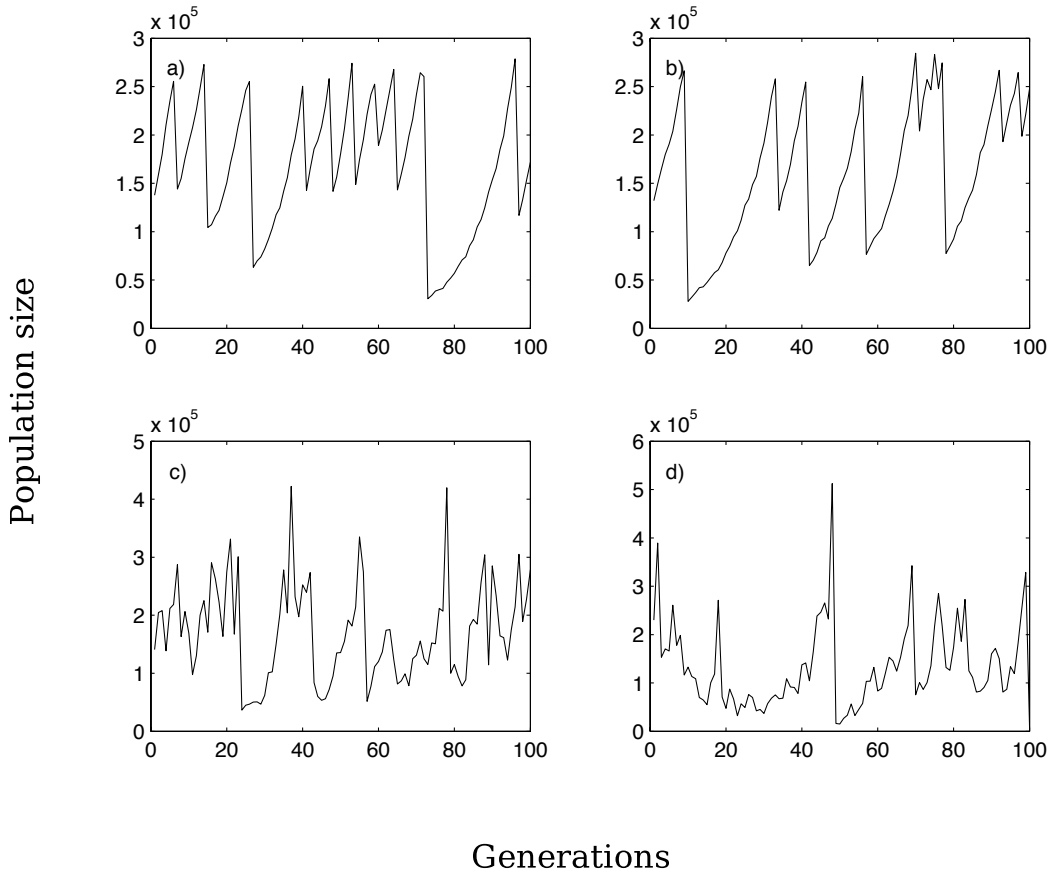


Figure 7: Examples of population size fluctuations achieved with the Ricker logistic growth model (Ricker 1954). a & b) low variance $\sigma^2 = 0.001$; c & d) high variance $\sigma^2 = 0.1$.

both high and low population size variance (Figure 8), although the latter is characterized by a sudden drop in neighbourhood size after seventy generations. In addition, for low to intermediate dispersal (d = 2, d = 4) the final neighbourhood sizes after 100 generations are lower than that of constant population size, given the same starting conditions and dispersal parameters. At a small starting neighbourhood size, subpopulation genetic differentiation is expected to be high, and when dispersal is low, this subpopulation genetic differentiation should be maintained by random genetic drift within local subpopulations. Neighbourhood sizes are most likely smaller since the probability for alternate alleles to become fixed in different subpopulations is greater, than in a constant population size scenario. This notion is supported by the observed reduction in heterozygosity under population size fluctuations (Figure 9).

Rapid decreases in neighbourhood size are observed for intermediate and large starting neighbourhood sizes under high population size variance and to an extent under low population size variance (Figure 8). Again, given low levels of dispersal the probability of alternate alleles becoming fixed, due to random genetic drift when population size is substantially reduced, is greater than when dispersal is greater. In general, population size fluctuations, and lower dispersal distances cause one to underestimate neighbourhood size, the degree of which is dependent on dispersal distance. In addition, the speed at which neighbourhood size is reduced under low levels of dispersal is dependent on the variance in population size fluctuations, with larger fluctuations causing faster reductions in inferred neighbourhood size. Finally, no apparent differences between neighbourhood sizes calculated from a fluctuating versus constant population are observed at large dispersal distances (d = 6). Deviations from this generalization are evident at an intermediate starting neighbourhood size, where neighbourhood size after 100 generations is substantially lower than in the constant population size scenario. Since each simulation is an independent stochastic event, deviations that are the result of the independent population size changes over time are expected. In particular, the population size change profile for the anomalous simulation was characterized by a large population crash after sixty generations, and a failure to recover from low numbers before the end of the simulation. This large population crash, would first decrease allelic diversity overall (Figure 9), but through random choosing of alleles for the generation subsequent to the crash allow for alternate alleles to become fixed in different positions on the lattice. This would have the overall result of reducing neighbourhood size; and preventing a subsequent recovery to normal levels, with a population size increase, before the end of the simulation. However, the general result for high dispersal, of limited effect, is intuitive since large dispersal distances will tend to homogenize, through gene flow, any local genetic differentiation that has occurred as a result of a decrease in population size. It is fascinating, however, that a mean parent-offspring distance of only 6 lattice units (less than 1.5% of the dimensions of the lattice) is sufficient for

Table 7: Relationship between dispersal parameter, *d*, and the equilibrium neighbourhood size for a continuous population of constant size. Mean neighbourhood size (and standard deviations) are presented for the last 200 generations, of a total of 1000 generations simulated, for three levels of dispersal, *d*, and three levels of starting population neighbourhood size, *Nb*.

	***d* = 2**	***d* = 4**	***d* = 6**
***Nb* = 10**	9.66 (1.0)	30.80 (4.10)	51.28 (20.58)
***Nb* = 25**	12.03 (1.69)	31.72 (5.02)	47.67 (22.55)
***Nb* = 50**	10.56 (1.13)	25.31 (6.82)	40.95 (10.13)

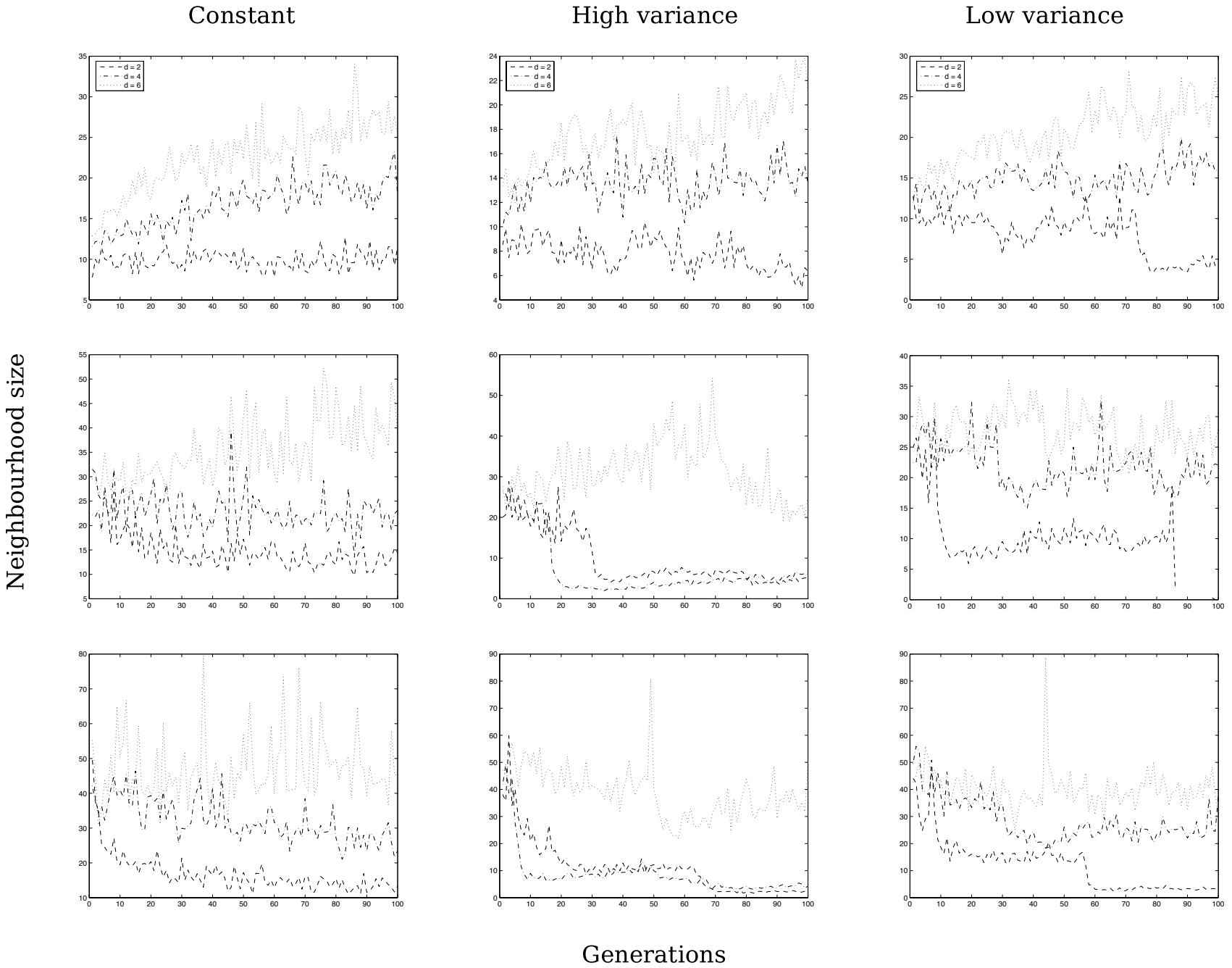


Figure 8: Changes in neighbourhood size over time under constant population size and fluctuating population size for three levels of dispersal, and three starting population neighbourhood sizes.

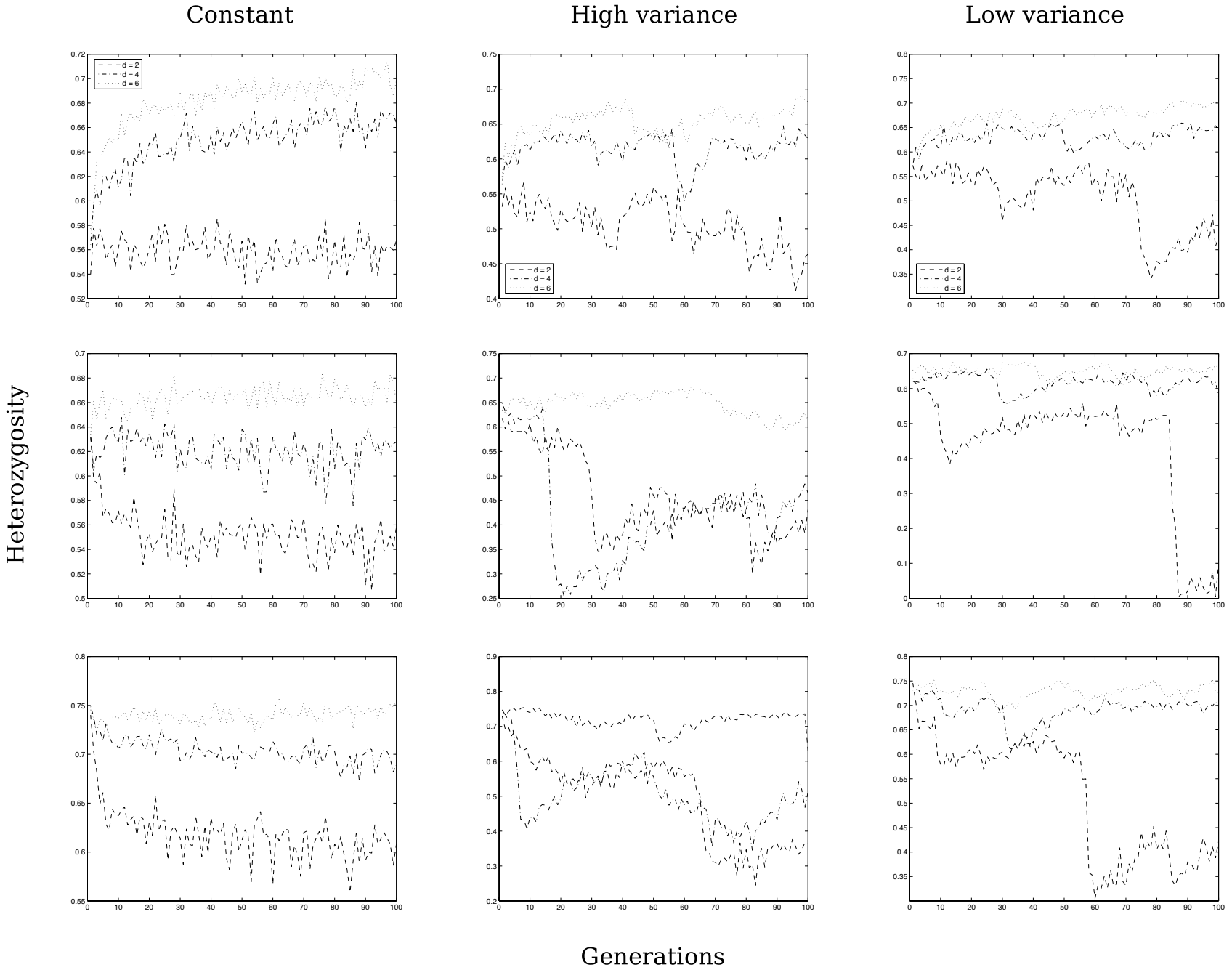


Figure 9: Changes in heterozygosity over time under constant population size and fluctuating population size for three levels of dispersal, and three starting population neighbourhood sizes.

population size fluctuations not to have any visible effect on the estimation of neighbourhood size.

Discussion

The major conclusion in this study is that the inference of demographic parameters from genetic data is highly susceptible to the assumptions of the demographic model used in analyzing the data. Thus far no analytical models are suitable for analyzing population genetic data from species with large annual population size fluctuations. We have shown using simulation modeling that the inference of neighbourhood size from microsatellite data obtained from such a species is dependent on both the degree of population size fluctuations, and on the level of dispersal. Theoretically, given a species with low dispersal ability, one would tend to underestimate neighbourhood size, as a result of local subpopulation differentiation. However, the challenge remains to interpret the data derived from *G. postica*, our focal species in this study.

Estimates of dispersal ability

The morphological estimates of dispersal as determined by wing loading give an indication of dispersal ability of *G. postica*. In Lepidoptera, there is a positive correlation between body mass and wing loading (Casey & Joos 1983), and generally a lower wing load translates into more energy-efficient flying. This relationship exists since the wing stroke frequency decreases with body size (Casey & Joos 1983). At the extreme of wing load in moths are the sphingid moths, which are considered the fastest flying group of insects (Matthews 1992), and are renowned for their hovering ability when drinking nectar from flowers on the wing. Sphingid moths are known to disperse and migrate long distances (Janzen 1984, Haber & Frankie 1989). The tobacco hawk moth (*Manduca sexta*), one of the largest sphingid species, has a mass which varies between 1 and 3g, wing length of 40-60mm and wing loadings of 7-15 N m^{-2} (Stevenson *et al.* 1995). *G. postica* is a moth of similar mass (males: 0.26 – 0.91g, females: 1.35 – 4.15g), yet has substantially higher wing load, the result of smaller wing surface area. Given such a high wing load, *G. postica* would need an extremely high wing stroke frequency for sustained flight, which would be energetically expensive. Efficient flying is necessary for *G. postica*, since the adult moths have no feeding mouth parts and rely on reserves from the pupal stage to sustain flight. Species with high wing stroke frequencies have flights that are irregular and characterized by erratic movements (Rydell & Lancaster 2000). Field observations of *G. postica* females, at light traps, support this notion (pers obs). Males on the other hand appear to be more efficient fliers, due to lower wing load. In addition, males begin warming up their wings with high frequency wing strokes soon after emergence from the cocoon

(pers obs). Rydell & Lander (2000) attribute the need for warming-up flight muscles to species with higher wing loads that need to achieve a higher thoracic temperature than species with lower wing loads. Given this observed difference between male and female *G. postica*, females may be more sedentary and males dispersive. The indirect evidence presented here, however, is problematic since wing load may not be sufficient to determine dispersal ability. For example, some small-bodied geometrid moths are known to have very low wing loads, yet are poor fliers (Rydell *et al.* 1997). Furthermore, non-feeding saturniid moths, a closely related family of lasiocampids, have been shown to be good dispersers (Waldbauer & Sternberg 1982), despite large body sizes and high wing loads. Furthermore, wing muscle mass may be a more important determinant of dispersal ability than wing load *per se*, and would therefore be important to consider in future studies. The evidence in support of low dispersal ability in *G. postica* is limited. Yet the absence in *G. postica* of a stability-enhancing fulcrum and spine between the hind- and forewings, characteristic of good fliers such as the Hawk moths, provides additional support for low dispersal ability in this species. In conclusion, indirect evidence of dispersal does suggest low vagility, yet these results would need to be confirmed with capture-mark recapture studies.

The potential for differences in male and female dispersive ability would certainly lead to differences in the inferred spatial genetic patterns from mitochondrial and nuclear loci. However, in this study little population genetic structuring is observed in both data sets. F_{ST} values for mtDNA (Table 4), are generally greater than that of microsatellites (Figure 6). This result would be expected if females were less vagile than males, yet population size fluctuations could reduce mtDNA diversity. Since organellar genes (mtDNA) have one quarter the population size of nuclear genes the loss of organellar genetic diversity under severe bottlenecks is greater (Wilson *et al.* 1985, Grant & Leslie 1993). In particular, Grant and Leslie (1993) note that contrary to Northern Hemisphere species, greater levels of diversity are often observed in nuclear genes than in mitochondrial genes from southern African taxa. The authors attribute this loss of mtDNA diversity to the frequent extinctions and recolonisations occurring in southern African populations as a result of drought and rainfall cycles. Thus the observed population genetic pattern in mtDNA may be the result of reduced female versus male dispersal coupled with the loss of diversity as a result of population size fluctuations.

Through simulations we have shown that population size fluctuations combined with low dispersal can lead to gross underestimates of neighbourhood size from microsatellite data, and thus substantial genetic structure. However, intermediate dispersal will tend to have no apparent effect on the inference of neighbourhood size. The microsatellite data generated in this study is

characterized by a few high frequency alleles, yet there is no obvious pattern of fixing of alternate alleles in different subpopulations. Rather, the data are representative of an essentially panmictic population with a very weak signal of isolation by distance. Thus we believe the data observed are the result of population fluctuations with intermediate levels of dispersal, whereby genetic diversity is homogenized across the distribution of the species. A potential problem with comparing simulation results to patterns observed in real data is interpreting the spatial scale at which the effects occur. For example, a level of dispersal less than 1.5% of the size of the lattice area is sufficient for genetic diversity to withstand the effects of population size fluctuations in a simulation, yet this is not easily translated to a species dispersal distance. In particular, making inferences of the geographic extent of subpopulations or eruptions in such a case would be futile. Rather, one should consider the spatial analysis results.

The test for spatial autocorrelation of alleles with geographic distance indicated a pattern not significantly different from a random association. These results are partly due to the inability for spatial autocorrelation statistics to incorporate potential sources of error in population genetic data (Slatkin & Arter 1991). The authors stressed that population genetic data are characterized by three sources of variance: sampling, stochastic and parametric. These sources of variance amount to basing inference on only a subset of samples, the stochasticity of the lineage sorting process, and averaging across loci with potentially divergent mutation rates. Although these criticisms of spatial autocorrelation methods are robust, we believe the patterns observed in our data are rather the result of the failure for equilibrium to be reached. Sokal & Wartenberg (1983) have shown that correlograms are established within 50 generations, and furthermore within 150 generations correlograms reach a quasi-stationary state, which persists for hundreds of generations (Epperson 1995). Given the little we know of population size fluctuations in *G. postica* it is unlikely that correlograms, indicative of isolation by distance, would have reached an equilibrium state. Hierarchical F-statistic analysis showed a substantial reduction in F_{ST} values beyond a geographical extent of 50km. Spatial regression of F_{ST} against distance however, combined with rudimentary estimates of density, indicated a natal dispersal distance of between 11 and 42m. However, since the regression of F_{ST} against distance is not significantly different from zero, this estimate is an absolute minimum dispersal distance. A less steep regression would imply larger neighbourhood sizes and thus larger dispersal distances. Whether, occasional long-distance dispersal or stepping-stone-effects with low dispersal distances can generate the observed patterns is questionable. Most likely, the observed patterns are the result of the combined effects of population size fluctuations, loci with low to intermediate mutation rates and low to intermediate levels of dispersal. Similar studies on continuously distributed cyclical species indicate little population genetic structuring. For example, the observed high levels of gene flow in the cyclic snowshoe hare (*Lepus americanus*) has

been attributed to a stepping-stone model of gene flow influenced by density cycles, where local bottleneck populations expand to previously unsuitable habitat and thus homogenize genetic diversity across the distribution (Burton *et al.* 2002). Another cyclic species that showed low genetic differentiation in a continuous distribution, the collared lemming, has its population genetic structure attributed to potential long-distance dispersal events (Ehrich *et al.* 2001). Indeed Burton *et al.* (2002) have noted the proliferation of high gene flow genetic signals derived from population genetic studies of cyclical species, and suggested this association may be worthy of further investigation. In this manuscript we report a similar occurrence in cyclical populations of African Wild Silk Moth. Through the use of simulations we show that neighbourhood size can be reduced, at low dispersal, yet cannot be increased as a result of population size fluctuations. Thus large neighbourhood sizes, and reduced population genetic structure in *G. postica*, does not appear to be the result of population size fluctuations. Though this observation would be dependent on the mutation rate of the loci considered. Loci with high mutation rates might be resistant to population size fluctuations, whereas loci with lower mutation rates may be more susceptible to increased homoplasy under population size fluctuations. Given lower levels of allelic diversity, the potential for the same allele to become fixed in different subpopulations, during population crashes, is greater than with higher levels of allelic diversity. This could result in the observed inference of high levels of gene flow in population genetic studies of cyclical species. We are currently investigating this process with simulation modeling.

One of the motivating factors for a population genetic study on *Gonometa postica* was the potential to guide the initiatives of the African Wild Silk industry in southern Africa. This study forms part of a larger genetic and ecological-based research programme that has been developed to gain an understanding of population cycles in this species. The purpose of the initial genetic survey was to determine the dispersal ability of *G. postica* and thus the appropriate scale at which future population dynamics studies should be conducted. Although, the genetic signal presented in this manuscript is weak, the reduction in F_{ST} at a pooling level greater than 50km, in the hierarchical F_{ST} analysis, provides a good indication of the extent of eruptions in this species. The addition of more microsatellite loci may provide greater resolution. In addition, temporal genetic sampling and analysis may assist in further evaluating this result. Since eruptions may result in an increased frequency of a locally rare allele, the presence of these rare alleles in adjacent regions in subsequent years may allow the inference of dispersal from the preceding eruption. We are currently investigating the potential for temporal rare allele frequencies to be informative in estimating dispersal. Knowledge of the dispersal ability of *G. postica* is crucial for the understanding of population cycles in this species, since it determines whether the species will persist in all regions given recent population crashes. The ability for relict populations to reseed regions where the

species has declined determines the long-term sustainability of the species and thus the long-term sustainability of a harvesting programme. Ultimately, dispersal estimates and the degree of spatial and temporal connectivity will be used to complement population dynamic modeling that is planned for this species. We look forward to a novel opportunity to incorporate genetic, ecological and modeling approaches to understanding the population biology of this species, and thereby providing harvesting recommendations to the Wild Silk Industry.

Acknowledgements

This work was funded by the Mellon Foundation Grant to J. W. H. Ferguson, P. Bloomer and W. Delport, and by a National Research Foundation grant to P. Bloomer (Gun: 2053653). The opinions and views presented in this article are however, not necessarily those of the National Research Foundation. Furthermore, we would like to thank Louis Hauman, Duncan McFadyen and E. O. Oppenheimer & Son for accommodation assistance during sampling. Thanks also to Fourie Joubert and deepthought (http://deepthought.bi.up.ac.za), a 64-node cluster, for computational support.

References

Arnaud JF, Madec L, Guiller A, Bellido A (2001) Spatial analysis of allozyme and microsatellite DNA polymorphisms in the land snail *Helix aspera* (Gastropoda: Helicidae). *Molecular Ecology*, **10**, 1563-1576.

Avise JC (2000) Phylogeography: The history and formation of species. Harvard University Press, Harvard.

Avise JC, Arnold J, Ball RM, Bermingham E, Lamb T, Neigel JE, Reeb CA & Saunders NC. (1987) Intraspecific phylogeography: The mitochondrial DNA bridge between population genetics and systematics. *Annual Review of Ecology & Systematics*, **18**, 489-522.

Beerli P & Felsenstein J. (1999) Maximum-likelihood estimation of migration rates and effective population numbers in two populations using a coalescent approach. *Genetics*, **152**, 763-773.

Beerli P & Felsenstein J. (2001) Maximum-likelihood estimation of a migration matrix and effective population sizes in n subpopulations by using a coalescent approach. *Proceedings of the National Academy of Sciences, USA*, **98**, 4563-4568.

Bowers RG, Begon M, Hodgkinson (1993) Host-pathogen population cycles in forest insects: Lessons from simple models reconsidered. *Oikos*, **67**, 529-538.

Burton C, Krebs CJ, Taylor EB (2002) Population genetic structure of the cyclic snowshoe hare (*Lepus americanus*) in southwestern Yukon, Canada. *Molecular Ecology*, **11**, 1689-1701.

Casey TM, Joos BA (1983) Morphometrics, conductance, thoracic temperature, and flight energetics of Noctuid and Geometrid moths. Physiological Zoology, **56**, 160-173.

Clement M, Posada D, Crandall KA (2000) TCS: a computer program to estimate gene genealogies. *Molecular Ecology*, **9**, 1657-1659.

Delport W (2005) Characterisation of six microsatellite loci in the African Wild Silk Moth (*Gonometa postica*, Lasiocampidae). Chapter 3, University of Pretoria thesis, Pretoria, South Africa.

Delport W (2005) Temporal and spatial distribution of African Wild Silk Moth, *Gonometa postica*, eruptions in southern Africa. Chapter 2, University of Pretoria thesis, Pretoria, South Africa.

Diniz-Filho JAF, Telles MPC (2002) Spatial autocorrelation analysis and the identification of operational units for conservation in continuous populations. *Conservation Biology*, **16**, 924-935.

Ehrich D, Jorde PE, Krebs CJ, Kenney AJ, Stacy JE, Stenseth NC (2001) Spatial structure of lemming populations (*Dicrostonyx groenlandicus*) fluctuating in density. *Molecular Ecology*, **10**, 481-495.

Epperson BK & Li T (1996) Measurement of genetic structure within populations using Moran's spatial autocorrelation statistics. *Proceedings of the National Academy of Sciences, USA*, **93**, 10528-10532.

Epperson BK (1995) Spatial distributions of genotypes under isolation by distance. *Genetics*, **140**, 1431-1440.

Excoffier L, Smouse P, Quattro J (1992) Analysis of molecular variance inferred from metric distances among DNA haplotypes: Application to human mitochondrial DNA restriction data. *Genetics*, **136**, 343-359.

Excoffier L (2004) Patterns of DNA sequence diversity and genetic structure after a range expansion: lessons from the infinite island model. Molecular Ecology, **13**, 853-864.

Fenster CB, Vekemans X, Hardy OJ (2003) Quantifying gene flow from spatial genetic structure data in a metapopulation of Chamaecrista fasciculata (Leguminosae). Evolution, **57**, 995-1007.

Freddi G, Bianchi Svilokos A, Ishikawa H, Tsukada M (1993) Chemical composition and physical properties of *Gonometa rufobrunnea* silk. *Journal of Applied Polymer Science*, **48**, 99-106.

Ginzburg LR, Taneyhill DE (1994) Population cycles of Forest Lepidoptera: A maternal effects hypothesis. *Journal of Animal Ecology*, **63**, 79-92.

Goudet J (1995) Fstat version 1.2: a computer program to calculate F-statistics. *Journal of Heredity*, **86**, 485-486.

Goudet J, De Meeüs T, Day AJ, Gliddon CJ (1994) The different levels of population structuring of the dogwhelk, *Nucella lapillus*, along the South Devon coast. In: *Genetics and Evolution of Aquatic Organisms* (ed. Beaumont AR), pp. 81-95. Chapman & Hall, London.

Grant WS, Leslie RW (1993) Effect of metapopulation structure on nuclear and organellar DNA variability in semi-arid environments of southern Africa. *South African Journal of Science*, **89**, 287-293.

Haber WA & Frankie GW (1989) A tropical hawkmoth community: Costa Rican dry forest Sphingidae. Biotropica, **21**, 155-172.

Hardy OJ, Vekemans X (1999) Isolation by distance in a continuous population: reconciliation between spatial autocorrelation analysis and population genetics models. *Heredity*, **83**, 145-154.

Hardy OJ & Vekemans X (2002) SPAGeDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes*, **2**, 618-620.

Hardy OJ & Vekemans X (2003) SPAGeDi 1.1: a program for *S*patial *P*attern *A*nalysis of *Ge*netic *Di*versity, User's manual.

Hastings A, Harrison S (1994) Metapopulation dynamics and genetics. *Annual Review of Ecology and Systematics*, **25**, 167-188.

Hudson RR (1990) Gene genealogies and the coalescent process. In: Oxford Surveys in Evolutionary Biology Volume 7 (eds. Futuyama D & Antonovics J), pp 1-43. Oxford University Press, Oxford.

Janzen DH (1984) Two ways to be a tropical big moth: Santa Rosa saturniids and sphingids. In: Oxford Surveys in Evolutionary Biology Volume 1 (eds), pp 85-140. Oxford University Press, Oxford.

Kimura M, Crow JF (1964) The number of alleles that can be maintained in a finite population. *Genetics*, **49**, 725-738.

Kingman JFC (1982) The coalescent. Stochastic Processes and their Applications, **13**, 235-248.

Knowles LL, Maddison WP (2002) Statistical phylogeography. *Molecular Ecology*, **11**, 2623-2635.

Knowles LL (2004) The burgeoning field of statistical phylogeography. *Journal of Evolutionary Biology*, **17**, 1-10.

Kuhner MJ, Yamato J, Felsenstein J (1998) Maximum likelihood estimation of population growth rates based on coalescent. *Genetics*, **149**, 429-434.

Kuhner MK, Yamato J, Beerli P, Smith LP, Rynes E, Walkup E, Li C, Sloan J, Colacurcio P, Felsenstein J (2004) LAMARC v 1.2.1. University of Washington, http://evolution.gs.washington.edu/lamarc.html.

Leblois R, Estoup A & Rousset F (2003) Influence of mutational and sampling factors on the estimation of demographic parameters in a “continous” population under isolation by distance. *Molecular Biology and Evolution*, **20**, 491-502.

Leblois R, Rousset F, Estoup A (2004) Influence of spatial and temporal heterogeneities on the estimation of demographic parameters in a continuous population using individual microsatellite data. *Genetics*, **166**, 1081-1092.

Loiselle BA, Sork VL, Nason J, Graham C (1995) Spatial genetic structure of tropical understory shrub, Psychotria officinalis (Rubiaceae). *American Journal of Botany*, **82**, 1420-1425.

Malécot (1951) Un traitement stochastique des problémes linéaires (mutation, linkage, migration)

en génétique des populations. *Annales de l'Université de Lyon A*, **14**, 79-117.

Matthews P (1992) (ed.) The Guinness Book of Records 1993. New York.

McGeoch MA, Veldtman R, Scholtz CH (2004) Feasibility of annual cocoon yield requirements. Report to Wild Silk Africa. University of Stellenbosch, South Africa.

Nielsen R, Wakeley JW (2001) Distinguishing Migration from Isolation: an MCMC Approach. *Genetics*, **158**, 885-896.

Ohta T & Kimura M (1973) A model of mutation appropriate to estimate the number of electrophoretically detectable alleles in a finite population. *Genetical Research*, **22**, 201-204.

Posada D & Crandall KA (2001) Intraspecific gene genealogies: trees grafting into networks. *Trends in Ecology and Evolution*, **16**, 37-45.

Pritchard JK, Rosenberg NA (1999) Use of unlinked genetic markers to detect population stratification in association studies. *American Journal of Human Genetics*, **65**, 220-228.

Rasband WS (1997) *ImageJ*, U. S. National Institutes of Health, Bethesda, Maryland, USA, http://rsb.info.nih.gov/ij/.

Ray N, Currat M & Excoffier L (2003) Intra-deme molecular diversity in spatially expanding populations. *Molecular Biology and Evolution*, **20**, 76-86.

Raymond M & Rousset (1995) GENEPOP (version 1.2): population genetics software for exact tests and ecumenicism. *Journal of Heredity*, **86**, 248-249.

Ricker WE (1954) Stock and recruitment. *J. Fish. Res. Board Can.*, **11**, 559–663.

Ritland K (1996) Estimators for pairwise relatedness and individual inbreeding coefficients. *Genetical Research Cambridge*, **67**, 175-185.

Roff DA & Bentzen P (1989) The statistical analysis of mitochondrial DNA polymorphisms: chi-square and the problem of small samples. *Molecular Biology and Evolution*, **6**, 539-545.

Rogers AR & Harpending HC (1992) Population growth makes waves in the distribution of pairwise genetics differences. *Molecular Biology and Evolution*, **9**, 552-569.

Rousset F (1997) Genetic differentiation and estimation of gene flow from F-statistics under isolation by distance. *Genetics*,**145**, 1219-1228.

Rousset F (2000) Genetic differentiation between individuals. *Journal of Evolutionary Biology*, **13**, 58-62.

Rydell J, Skals N, Surlykke A & Svensson M (1997) Hearing and bat defence in geometrid winter

moths. *Proceedings of the Royal Society of London B*, **264**, 83-88.

Rydel J, Lancaster WC (2000) Flight and thermoregulation in moths were shaped by predation from bats. *Oikos*, **88**, 13-18.

Schneider S, Roessli D & Excoffier L (2000) ARLEQUIN version 2: A software for population genetic data analysis. Genetics and Biometry Laboratory, University of Geneva, Geneva, Switzerland.

Slatkin M & Arter HE (1991) Spatial autocorrelation methods in population genetics. *The American Naturalist*, **138**, 498-517.

Sokal RR, Oden NL (1978) Spatial autocorrelation in biology. 1. Methodology. *Biological Journal of the Linnean Society*, **10**, 199-228.

Sokal RR & Wartenberg DE (1983) A test of spatial autocorrelation analysis using an isolation-by-distance model. *Genetics*, **105**, 219-237.

Stevenson RD, Corbo K, Baca LB & Le QD (1995) Cage size and flight speed of the tobacco hawkmoth *Manduca sexta*. Journal of Experimental Biology, **198**, 1665-1672.

Tajima F (1989) Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics*, **123**, 585-595.

Turchin (2003) Complex Population Dynamics: a theoretical/empirical synthesis. *Monographs in Population Biology*, **35**. Princeton University Press, Princeton.

Vekemans X & Hardy OJ (2004) New insights from fine-scale spatial genetic structure analyses in plant populations. *Molecular Ecology*, **13**, 921-936.

Veldtman R (2004) The ecology of southern African wild silk moths (*Gonometa* species, Lepidoptera: Lasiocampidae): consequences for their sustainable use. University of Pretoria PhD thesis.

Veldtman R, McGeoch MA & Scholtz CH (2002) Variability in cocoon size in southern African wild silk moths: implications for sustainable harvesting. *African Entomology*, **10**, 127-136.

Veldtman R, McGeoch MA & Scholtz CH (2004) The parasitoids of southern African wild silkmoths (Lepidoptera). *African Entomology*, **12**, 117-122.

Wahlund S. 1928. Zusammensetzung von Populationen und Korrelation-serscheinungen von Standpunkt der Verebungslehre aus betrachtet. *Hereditas*, **11**, 65-106.

Waldbauer GP & Sternburg JG. (1982) Long mating flights by Hyalophora cecropia. *J. Lepid. Soc.*, **36**, 154-155.

Weir BS & Cockerham CC (1984) Estimating F-Statistics for the analysis of population structure. *Evolution*, **38**, 1358-1370.

Wilson AC, Cann RL, Carr SM, George M, Gyllensten UB, Helm-Bychowski KM, Higuchi RG, Palumbi SR, Prager EM, Sage RD & Stoneking M (1985) Mitochondrial DNA and two perspectives on evolutionary genetics. *Biol. J. Linn. Soc.*, **26**, 375-400.

Wright S (1931) Evolution in Mendelian populations. *Genetics*, **16**, 97-159.

Wright S (1940) Breeding structure of populations in relation to speciation. *American Naturalist*, **74**, 232-248.

Wright S (1943) Isolation by distance. *Genetics*, **28**, 114-138.

Wright S (1969) *Evolution and the Genetics of Populations.* Vol 2. *The Theory of Gene Frequencies*. University of Chicago Press, Chicago.

Chapter 5

Temporal and spatial genetic patterns in the African Wild Silk Moth (*Gonometa postica*) and implications for cyclical population dynamics

"So, if we can observe many genetic variations all of which can be assumed to be the result of the same forces, then the distribution of those variations can be used to estimate those forces."

Richard C. Lewontin (2002)

Temporal and spatial genetic patterns in the African Wild Silk Moth (*Gonometa postica*) and implications for cyclical population dynamics

Wayne Delport, J. Willem H. Ferguson & Paulette Bloomer

[1]Molecular Ecology and Evolution Programme, Department of Genetics, University of Pretoria, Pretoria, 0002, South Africa
[2]Centre for Environmental Studies, Department of Zoology and Entomology, University of Pretoria, Pretoria, 0002, South Africa

Abstract

The African Wild Silk Moth (*Gonometa postica*) exhibits large inter-annual population size changes. Previous studies indicated that the inference of dispersal ability in this species was complicated by the effect of population size fluctuations. In this manuscript we genotype samples collected in three successive years for six microsatellite loci. These data are used to estimate changes in effective population size and migration. Theoretically, migration could be tracked by changes in frequencies and occurrences of rare alleles when temporal samples are analysed. However, the results presented here show little changes in the spatial and temporal distribution and frequency of alleles. We suggest these results are due to the decrease in eruptive populations observed in the species over the temporal period sampled. Further temporal sampling, that includes population eruptions, would allow the inference of dispersal in this species.

Keywords: temporal samples, population cycles, effective population size, continuously distributed

Introduction

The African Wild Silk Moth, *Gonometa postica*, is a species than undergoes large inter-annual population size fluctuations (Veldtman 2004, Delport 2005 Chapter 4). Using simulations, Delport (2005 Chapter 4) showed that population size fluctuations in a continuously distributed species have the potential to generate increased population genetic structure, where demes experience local genetic drift and thus become fixed for alternate alleles (Wright 1940). This potential for population size fluctuations to generate population structure is only evident, however, when dispersal is low. In fact, a dispersal distance of more than 1% of the distribution of a species results in no significant effect of population size fluctuations on spatial genetic pattern versus that of a population with constant size (Delport 2005 Chapter 4). This result is contradictory to the high levels of gene flow inferred from several species with cyclical dynamics (Ehrich *et al.* 2001, Burton *et al.* 2002, Delport 2005 Chapter 4). Burton *et al.* (2002) attribute inferred high levels of gene flow in snowshoe hare (*Lepus americanus*) to a stepping-stone model of gene flow influenced by density cycles, where local bottleneck populations expand to previously unsuitable habitat and thus homogenize genetic diversity across the distribution. The high level of gene flow in collared lemming is potentially explained by long-distance dispersal events (Ehrich *et al.* 2001). However, Delport (2005 Chapter 4) found little indirect evidence in support of long-distance dispersal in a species of moth that survives only a few days, has low flying efficiency and lacks feeding mouthparts. Since population structure may be homogenised by population size fluctuations we consider an alternate method, using temporal genetic sampling, to identify the level of dispersal in this moth species.

Temporal genetic sampling has been shown to be useful in the estimation of effective population size (Krimbas & Tsakas 1971, Nei & Tajima 1981, Pollak 1983, Waples 1989), a measure of particular interest to conservation biologists. In a Wright-Fisher model of population size N, the theoretical variance in allele frequencies can be calculated given a starting allele frequency, and thus subsequently compared to the observed variance in allele frequencies, to estimate the effective population size N_e (Ewens 1979). These moment-based estimators, however, lack precision and typically

include confidence intervals that include infinity (Luikart *et al.* 1999, Berthier *et al.* 2002). Recently moment-based estimators have been replaced by maximum likelihood estimators (Williamson & Slatkin 1999, Anderson *et al.* 2000, Wang 2001, Berthier *et al.* 2002, Beaumont 2003, Wang & Whitlock 2003) that have the advantage of utilizing the full distribution of allele frequencies, and not simply summary statistics as in previous methods. In addition, the likelihood-based methods have allowed for changes to the underlying statistical model, such that joint estimation of N_e and migration in a stable population model (Wang & Whitlock 2003), or of N_e and population size changes (Beaumont 2003) can be achieved. Although temporal analysis of genetic data has previously been limited to rapidly mutating viruses (Drummond *et al.* 2002, Fu 2001) and bacteria (Falush *et al.* 2001), or to samples spaced over relatively long periods (Williamson & Slatkin 1999, Wang & Whitlock 2003, Jehle *et al.* 2001, Beaumont 2003), some studies have analysed temporal samples collected in successive years (Begon *et al.* 1980, Wang 2001, Beaumont 2003). These short-term studies, however, are restricted to insects with short generation times, such that 1-2 generations have elapsed between temporal samples. Insects, with short generation times, are particularly suitable for temporal genetic sampling since multiple generations can pass within the duration of a typical study. In addition, several insect species are characterized by large population size fluctuations (see Vandewoestijne *et al.* 1999, Ibrahim 2001, Bjornstad *et al.* 2002, Cooper *et al.* 2003, Turchin 2003, Delport 2005 Chapter 4 for examples), which have the potential to cause substantial changes in allele frequencies over short temporal periods.

Temporal allele frequency changes can provide information on migration patterns if a deme containing a rare allele experiences an eruption, and the rare allele is found in an alternate deme in the subsequent generation. Estimating the migration rate between these temporally spaced demes may allow the inference of dispersal. We use temporal genetic sampling combined with likelihood methods to estimate migration both within and between sampling periods, and changes in effective population size, N_e, over the sampling period. The principal aim of this manuscript is to infer dispersal patterns of the African Wild Silk Moth using temporally spaced genetic samples. Our results indicate little change in the spatial distribution of allele frequencies over the time period considered. We believe these results are due to the observed decrease in *G*.

postica eruptions (Delport 2005 Chapter 2), and we further suggest that long-term genetic monitoring of these populations will assist in identifying levels of dispersal in this species.

Materials and methods

Genetic sampling and laboratory methods

We collected pupae (cocoons) from *Acacia erioloba* and *Acacia mellifera* trees during the species' diapause in June 2002, July 2003 and July 2004 across the core outbreak region for the species (Veldtman *et al.* 2002, Delport 2005 Chapter 2). Adult moths were allowed to emerge in the laboratory and were frozen at -20°C for subsequent DNA extraction. We sampled a total of 206, 53 and 52 individuals in 2002, 2003 and 2004 respectively. Since apparent extinction of several local populations occurred in 2003 and 2004 (Delport 2005 Chapter 2) it was not possible to achieve the same sample sizes, and number of sampling localities (Figure 1) as in the first year. Total genomic DNA was extracted using a Qiagen Dneasy extraction kit and all individuals were genotyped for six polymorphic microsatellite loci (Gon6, Gon60, Gon65, Gon55, Gon107, Gon120). Details of protocols for microsatellite development, amplification conditions and summary statistics for loci are presented in Delport (2005 Chapter 3).

Statistical analysis methods

We conducted preliminary analysis of temporal allele frequencies by plotting distributions of allele frequencies derived from the three years of sampling. These distribution plots allowed for a visual inspection of changes in allele frequencies over the sampling period. Samples were grouped into localities based on geographic proximity (Figure 1 inset) and on the observed persistence of eruptions over the three-year sampling period (Delport 2005 Chapter 2). The practice of grouping sampling sites into demes may be problematic since the underlying distribution of the species appears continuous with spatial variation in local densities (pers obs). Grouping of localities can generate spurious results, such as the Wahlund effect (Wahlund 1928), that results in a deficiency of observed heterozygosity within populations. This effect is, however, only applicable when populations which have an underlying population

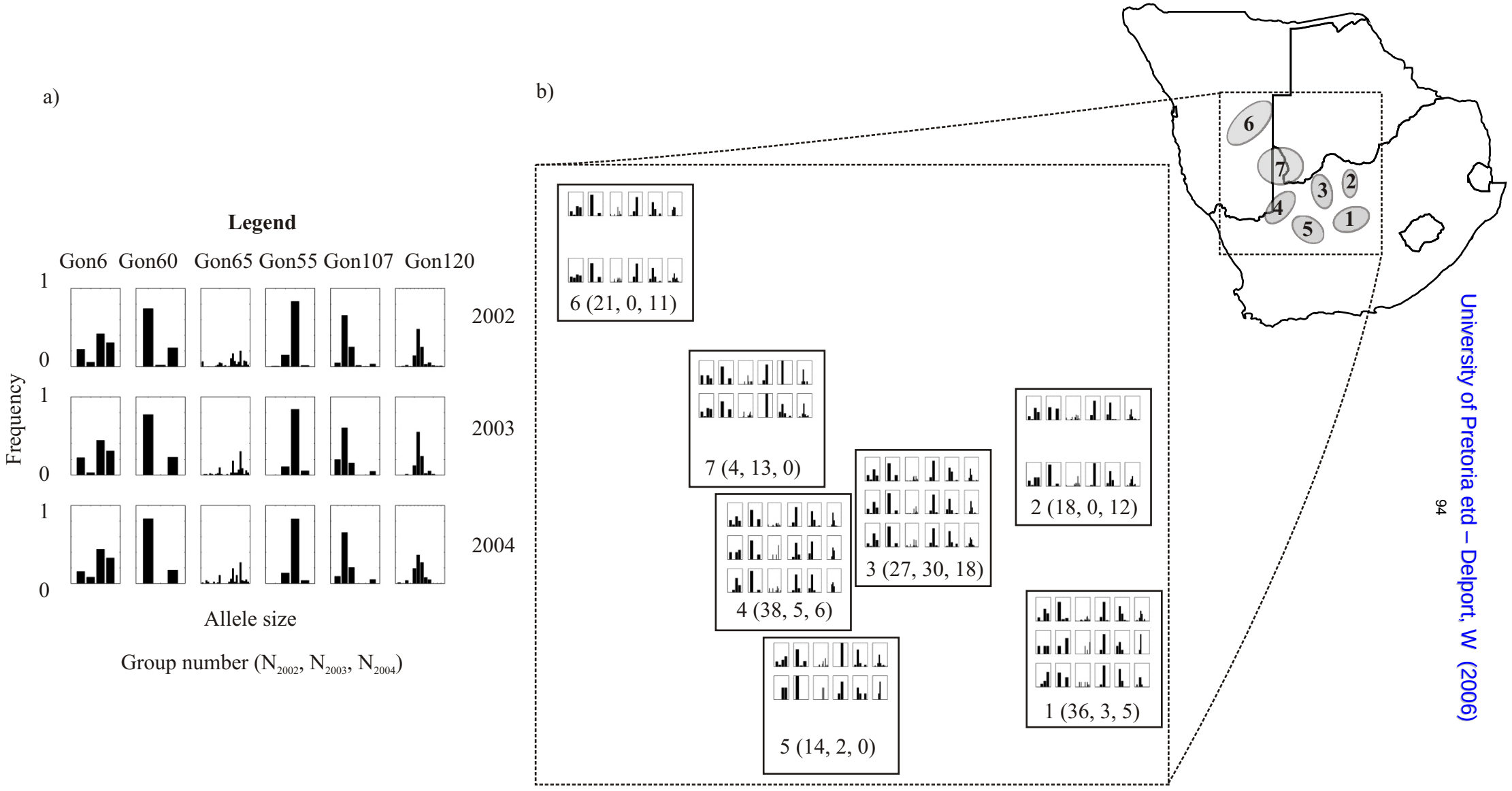


Figure 1: Temporal and spatial distribution of microsatellite allele frequencies. The allele frequency distribution a) for all localities, and b) per locality is represented. Figure 1a also serves as a legend for allele frequency distributions in 1b. Alleles are ordered in size from smallest to largest on the x-axis, with frequency on the y-axis. Locality groupings in subsequent analysis are indicated as gray polygons. Note locality groups 4 and 7 were combined for all subsequent analysis. N_x = sample size in year x. Sample sizes represented here are not consistent with those reported in the methods since several isolated individual samples could not be combined into the locality groups shown in this figure.

genetic structure are combined. Since there is little population genetic structure within *G. postica* (Delport 2005 Chapter 4) we consider the lumping of sampling sites into demes to be acceptable. Clearly, the analysis of continuously distributed data under an island model is not ideal, yet given that there are no alternative genealogical methods for the estimates of migration, we followed this approach. Some support for using an island model to infer gene flow in an apparently continuously distributed species is provided by temporal sampling. *G. postica* does not have overlapping generations over years, and thus temporal sampling of populations is analogous to an island model, where demes are temporally structured. The subsequent analysis comprised three approaches.

Firstly, we used a chi-square permutation procedure (Roff & Bentzen 1989) to determine whether the spatial distribution of allele frequencies at localities in each of the three years was significantly different from that expected at random. The method is justified for n x n contingency tables with low frequencies in cells, and for which there are no exact solutions available (Roff & Bentzen 1989).

Secondly, we used migrate-n (Beerli & Felsenstein 1999, 2001) to estimate migration rates between grouped sampling localities from 2002. Localities were grouped based on geographical proximity as before, yet localities 4 and 7 were combined since the sample size of locality 7 was insufficient for this analysis. Only localities for which there were > 5 samples in any year (mean = 20, sd = 11.45) were included in the analysis. The purpose of a subsequent temporal analysis was to identify whether single or multiple populations from 2002 could have sourced eruptions in 2003, and 2004 respectively. We used migrate-n (Beerli & Felsenstein 1999, 2001) to estimate migration rates between populations in successive sampling years. Only localities with sufficiently large sample sizes in 2003 and 2004 were used (2003: locality 3; 2004: locality 2 & 3). In the temporal migrate-n analysis we used a restricted custom migration model (Table 1) that only estimated one-way migration between populations in successive years. This migration model reduced the number of parameters to be estimated from the data thus ensuring the data were not over-analyzed. The use of migrate-n to identify temporal migration rates is questionable since the software does not model recruitment, however, this analysis would still

Table 1: Custom migration model enforced in migrate-n such that only migration between populations in successive years is estimated. This custom migration model was used such that the number of parameters estimated from the data was reduced from 64 to 25. Estimates of population size (θ) are shown along the diagonal and migration estimates above and below the diagonal. Migration from column to row is indicated in each cell where; * = no restriction, m = average migration across all demes (as per Beerli 2004), Y = year sampled, d = deme as in Figure 1.

		Migration from:							
	Y		**2002**	**2002**	**2002**	**2002**	**2002**	**2003**	**2004**	**2004**
		d	**1**	**2**	**3**	**4&7**	**5**	**3**	**2**	**3**
Migration to:	**2002**	**1**	*	m	m	m	m	m	m	m
	2002	**2**	m	*	m	m	m	m	m	m
	2002	**3**	m	m	*	m	m	m	m	m
	2002	**4&7**	m	m	m	*	m	m	m	m
	2002	**5**	m	m	m	m	*	m	m	m
	2003	**3**	*	*	*	*	*	*	m	m
	2004	**2**	*	*	*	*	*	*	*	m
	2004	**3**	*	*	*	*	*	*	m	*

provide some information of the degree of patch consistency over time (pers com. Peter Beerli). All migrate-n analyses were run with maximum-likelihood as the search strategy with ten short chains (50000 genealogies sampled, 1000 recorded) and three long chains (500000 genealogies sampled, 10000 recorded), and a burn-in of 10000 genealogies per chain. Furthermore, in order to check for consistency, migrate runs were repeated with starting parameters set at the maximum-likelihood estimates of the previous run.

Finally, since we observed changes in abundance between the three sampling years we used the temporal data to estimate changes in effective population size (Beaumont 2003). We used the software tmvp (Beaumont 2003) to obtain a maximum likelihood estimate of change in effective population size given the temporal microsatellite data. Preliminary runs of tmvp were conducted to determine the appropriate scale for the estimation of effective population size. We furthermore divided the analysis into two separate searches. Firstly, we used all the data from 2002 (206 individuals), 2003 (53 individuals) and 2004 (52 individuals), and estimated the likelihood surface of change in effective population size across six generations, given that *G. postica* cycles through two generations per annum (Hartland-Rowe 1992). Secondly, since the sample size from the first year (2002) was substantially larger than in subsequent years, we randomly sampled 50 individuals from the 2002 data. These data were combined with that from subsequent years to account for the effects of sample size in estimating change in effective population size. For both tmvp analyses the following importance sampling search options were used. Forty-thousand updates were proposed at a parameter step size of 0.15, and a thinning interval of 10; thus producing 4000 calculations of the likelihood given the observed data. Maxit, the size of the importance sample was set to 500. Preliminary analysis indicated that the likelihood surface for the first analysis should be estimated at a scale of 1000 individuals on each axis, whereas for the second, a scale of 5000 individuals per axis was most appropriate.

Results

Spatial and temporal distribution of allele frequencies

No clear changes in allele frequency are evident on either a spatial or temporal time scale (Figure 1). In general within a single sampling period, localities have one to two high-frequency alleles that are distributed throughout the range of the species (Figure 1). The observed spatial distributions of alleles for each locus were not significantly different from random as determined with permutations of the chi-square test (Table 2). These results provide support for the lumping of localities within years. This allelic distribution pattern is evident in all three sampling periods, where the high frequency alleles are maintained over time (Figure 1b). However, overall allelic diversity is reduced over time, as is evident in the loss of rare alleles (Figure 1a), particularly in Gon60, Gon65, Gon107 and Gon120. This loss of allelic diversity is attributable to one of two factors; the fewer number of eruptions found subsequent to 2002 and thus indicative of a population crash (Delport 2005 Chapter 2), or directly related to the former; the reduced sample sizes of 2003 and 2004.

Maximum likelihood estimation of demographic parameters

Maximum likelihood estimates (MLE's) of migration indicate the relative contribution of immigration versus mutation in generating new alleles within demes to be generally large (Table 3, Figure 2). Maximum likelihood estimates, however, should not be evaluated without consideration of the confidence limits surrounding such estimates. Confidence limits on the MLE taken at 2 log likelihood units below the maximum likelihood estimate (Edwards 1972) indicated little variation around the MLE and thus greater confidence in this estimate (Table 2). Migration estimates from a second search with the starting parameters set to the MLE of the previous run were in agreement with results from the first search. Very large migration estimates occur between locality 6, and each of localities 4&7 and 3. Lower estimates of migration however occur between adjacent localities 1 & 5, 3 & 5, 4&7 & 3, and 4&7 & 5. These results are generally in contradiction to what one might expect from an isolation by distance model, where an increase in geographic distance is accompanied by an increase in genetic distance (Wright 1943). The estimates of migration provided are given in terms of the contribution that immigration makes to the occurrence of

Table2: The spatial distribution of alleles for each of the six microsatellite loci. The results presented are from 1000 permutations of the chi-square test (Roff & Bentzen 1989) and represent the probability that the observed distribution of alleles among localities is significantly different from random. The Bonferroni corrected rejection level for multiple comparisons is 0.002. χ^2 = observed chi-square, *P* = proportion of random permutations with a χ^2 greater than that observed.

Locus	**Year**	χ^2	***P***
Gon6	2002	6.95	0.980
	2003	3.16	0.990
	2004	8.08	0.819
Gon60	2002	17.09	0.147
	2003	6.16	0.157
	2004	2.48	0.700
Gon65	2002	79.22	0.963
	2003	75.67	0.012
	2004	55.55	0.708
Gon55	2002	10.27	0.915
	2003	7.01	0.521
	2004	6.57	0.622
Gon107	2002	27.72	0.286
	2003	8.03	0.845
	2004	7.77	0.924
Gon120	2002	33.33	0.829
	2003	8.87	0.899
	2004	6.54	0.991

new alleles within a subpopulation/deme. However, it is useful to consider the effective number of migrants, *4Nm*. In this case simply multiplying the estimates of *M* by θ provides an estimate of the effective number of migrants (Beerli 2004) and ranges from less than 1 to approximately 12 (Table 3).

The 2002 migration estimates were inconsistent with the geographical location of samples. In general, higher levels of migration were observed between distant localities than between adjacent localities (Figure 2, Table 3). Therefore, we were unable to combine adjacent localities for the temporal analysis of migration. Rather we identified locality groups in the subsequent years in which there were sufficient sample sizes, and tested for temporal migration between these localities and adjacent localities from previous years. Therefore, the localities 1-5 were included from 2002, locality 3 in 2003 and localities 2 and 3 from 2004. In this way the number of parameters to be estimated was reduced, and localities with small sample sizes removed from the analysis. Smaller sample sizes for localities in successive years are indicative of small local population sizes, and not insufficient sampling, since search effort was consistent across localities and years. The temporal analysis of migration, in general, indicates high levels of gene flow (Figure 3, Table 4). Considering only the samples collected in 2002 and 2003, it appears that two of the five demes (4&7, 1) contributed most to the only deme sampled in 2003. Similarly, considering the results from 2002 and 2004, three of the five demes from 2002 appear to have contributed most to the eruption in 2004. In general, the temporal migration results indicate substantial gene flow across the range of the species. Confidence limits on these estimates of temporal migration were again small (Table 4) and a second run of migrate-n with the MLE's as start parameters resulted in similar estimates.

The analysis of change in effective population size using all the data collected in 2002 indicated a substantial change as inferred from the MLE of ancestral and recent effective population sizes (Figure 4a). The MLE (L = -323.4971) occurs at ancestral (AN_e) and recent (RN_e) effective population sizes of approximately 863 and 169 individuals, respectively (Figure 4a). The 95% confidence limit for the MLE shown is at –325.5. Evaluation of this confidence limit indicates that there is little confidence in the MLE, with AN_e and RN_e ranging from 300-1000 and 100-1000 respectively. When

Table 3: Maximum likelihood estimates of migration between 2002 demes (Figure 1) and θ ($4N\mu$, N = population size, μ = mutation rate) for each deme, as estimated with migrate-n (Beerli & Felsenstein 1999, 2001). Both M, the relative contribution of immigration versus mutation in generating new alleles ($M = m/\mu$, where m = immigration rate, μ = mutation rate) and $4Nm$, the effective number of migrants (N = population size, m = migration rate), are presented. *MLE*'s (maximum likelihood estimates) of M are shown for each deme/subpopulation pair, with $4Nm$ in parenthesis. MLE's of θ are boxed and shown along the diagonal. Table is read as immigration from column to row. Migration rates shown on Figure 2 are in bold. U = Upper 95% percentile of the estimated likelihood surface, L = lower 95% percentile of the estimated likelihood surface.

Migration from deme:

Migration to deme:		**1**	**2**	**3**	**4&7**	**5**	**6**
1	*MLE*	*0.47*	8.61 (4.1)	5.42 (2.6)	7.15 (3.4)	9.78 (4.7)	**18.72 (8.9)**
	L	*0.44*	7.57	4.61	6.20	8.67	**17.15**
	U	*0.51*	9.75	6.33	8.19	10.99	**20.37**
2	*MLE*	7.03 (3.1)	*0.45*	6.80 (3.0)	9.36 (4.2)	5.39 (2.4)	**14.5 (6.5)**
	L	6.14	*0.40*	5.92	8.33	4.61	**13.22**
	U	8.00	*0.49*	7.76	10.48	6.25	**15.90**
3	*MLE*	6.26 (2.3)	9.41 (3.5)	*0.37*	**19.1 (7.1)**	7.70 (2.9)	**31.9 (11.9)**
	L	5.35	*8.27*	*0.34*	**17.48**	6.67	**29.74**
	U	7.28	10.64	*0.41*	**20.85**	8.82	**34.10**
4&7	*MLE*	9.27 (4.3)	8.77 (4.1)	**18.1 (8.4)**	*0.47*	6.55 (3.1)	**16.49 (7.7)**
	L	8.25	7.77	**16.59**	*0.43*	5.69	**15.10**
	U	10.39	9.85	**19.58**	*0.50*	7.49	**17.96**
5	*MLE*	**11.1 (2.9)**	9.6 (2.5)	**11.0 (2.9)**	**14.0 (3.7)**	*0.26*	**11.0 (2.9)**
	L	**9.74**	8.29	**9.58**	**12.39**	*0.23*	**9.57**
	U	**12.66**	10.99	**12.49**	**15.67**	*0.30*	**12.46**
6	*MLE*	**20.8 (7.5)**	**17.6 (6.3)**	**34.6 (13)**	**25.3 (9.2)**	**12.4 (4.5)**	*0.36*
	L	**18.96**	**15.95**	**32.24**	**23.25**	**11.01**	*0.32*
	U	**22.66**	**19.35**	**37.02**	**27.32**	**13.86**	*0.41*

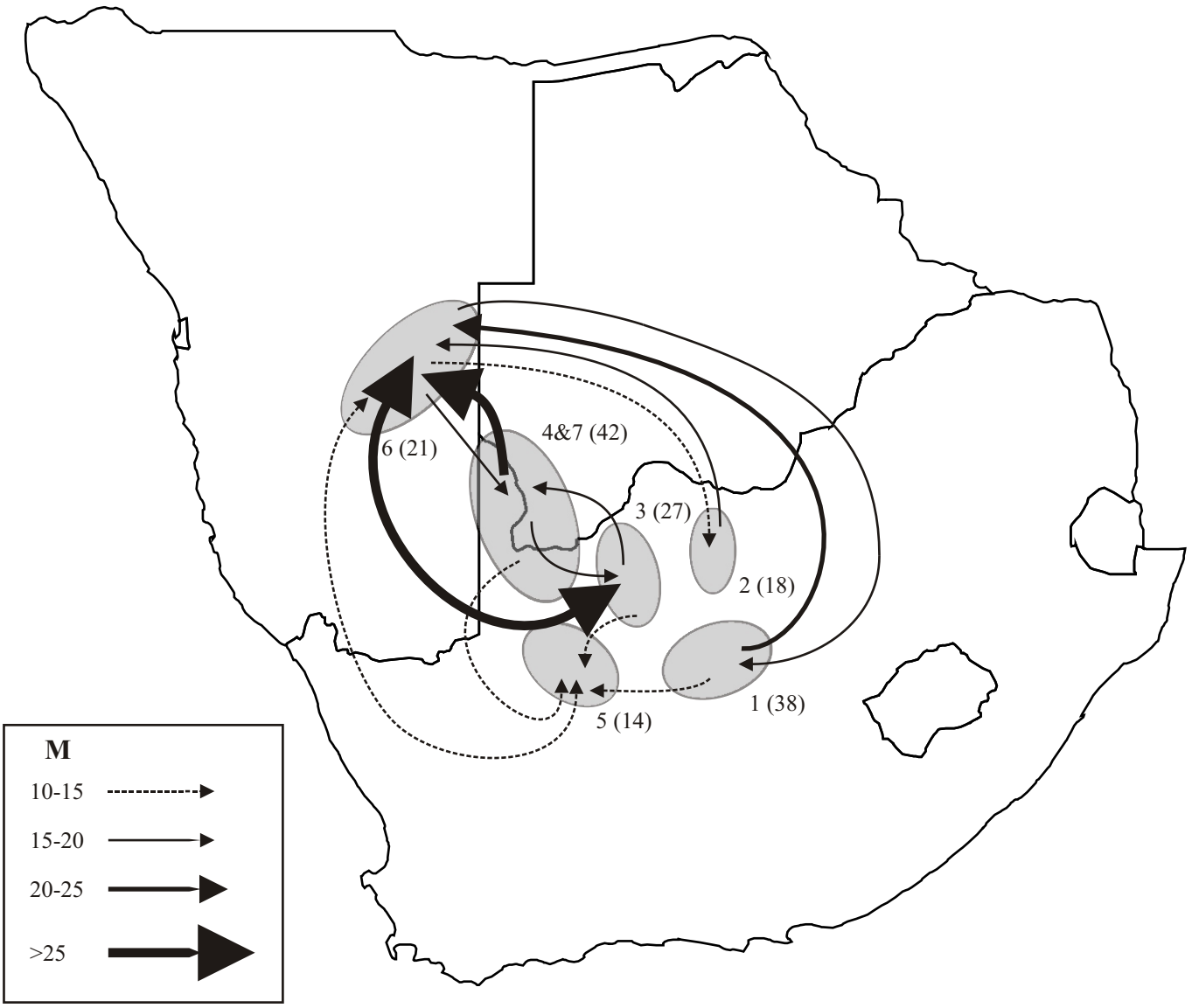


Figure 2: Maximum likelihood estimates of migration (from 2002 samples) using migrate-n (Beerli & Felsenstein 1999, 2001). The relative importance of migration versus mutation in generating new alleles in demes is presented as M, where $M = m/\mu$ (m = immigration rate, μ = mutation rate). Only M values > 10 are presented, whereas remaining immigration levels are shown in Table 2.

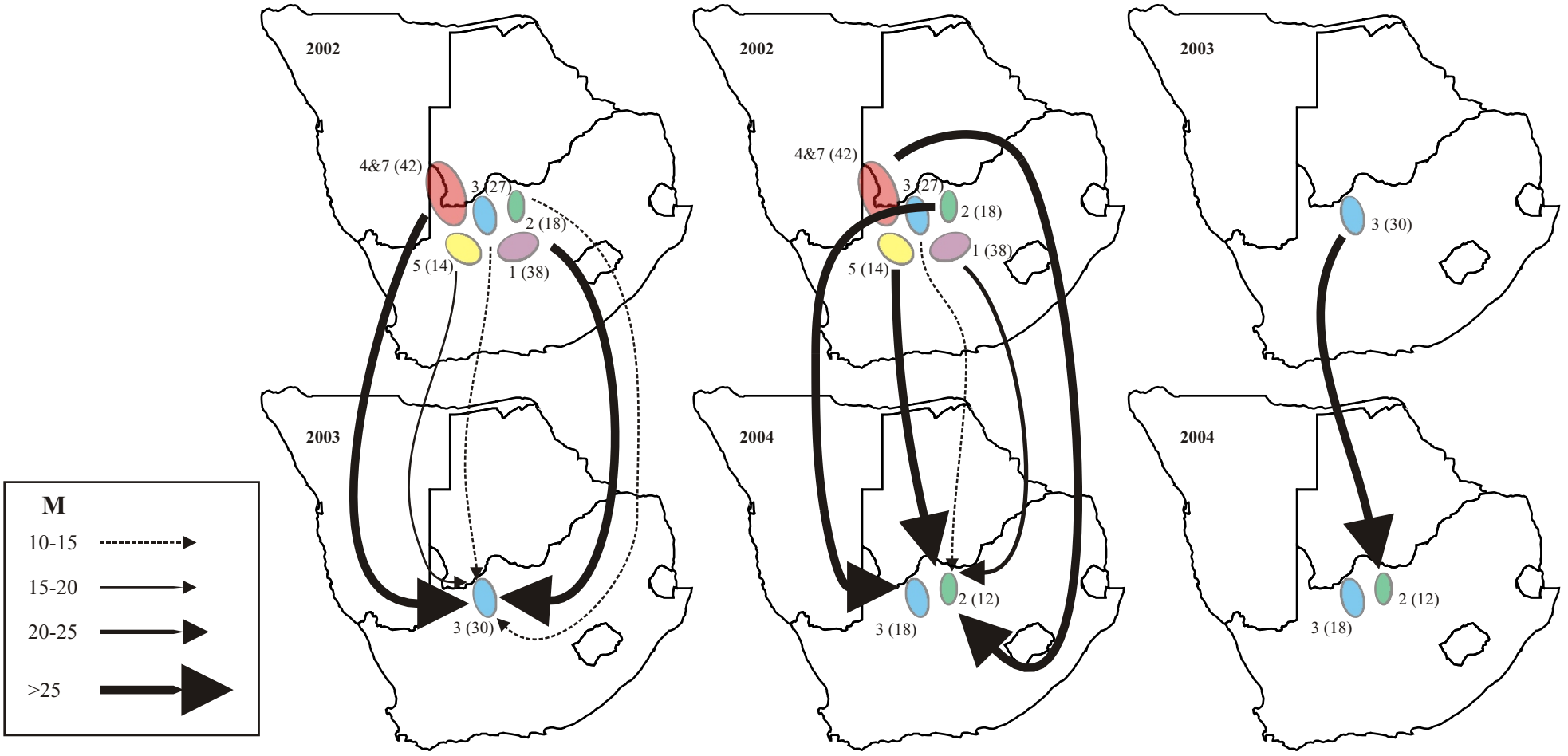
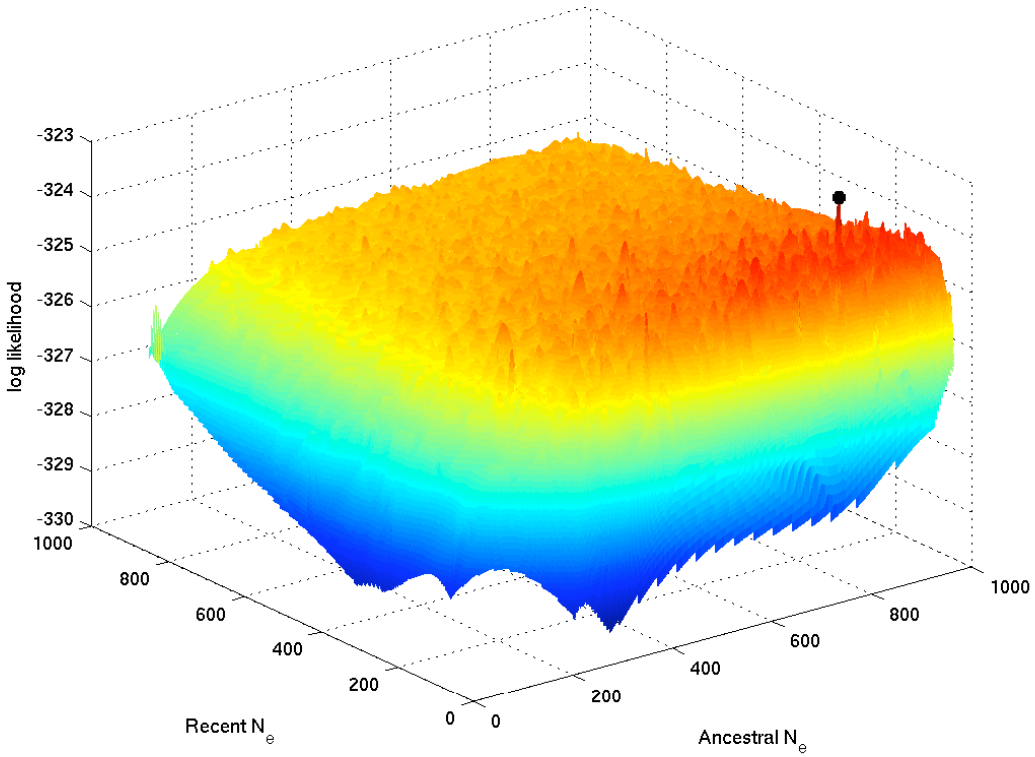


Figure 3: Temporal estimates of migration, $M = m/\mu$ (m = immigration rate, μ = mutation rate), between demes sampled in successive years. Only values greater than 10 are presented, whereas remaining immigration levels are shown in Table4.

Table 4: Maximum likelihood estimates of temporal migration and θ (boxed), as estimated with migrate-n (Beerli & Felsenstein 1999, 2001). Only migration between demes (d) collected in successive years (Y) was estimated so as to reduce the number of parameters estimated. Both *M*, the relative contribution of immigration versus mutation in generating new alleles ($M = m/\mu$, where m = immigration rate, μ = mutation rate) and *4Nm*, the effective number of migrants (N = population size, m = migration rate), are presented. *MLE*'s (maximum likelihood estimates) of *M* are shown for each deme/subpopulation pair, as is *4Nm*. MLE's of θ are boxed. Table is read as immigration from column to row. Migration rates shown on Figure 4 are in bold. *U* = Upper 95% percentile of the estimated likelihood surface, *L* = lower 95% percentile of the estimated likelihood surface, *N* = not estimated.

			Migration from deme:							
	Y	**d**		**2002 1**	**2002 2**	**2002 3**	**2002 4&7**	**2002 5**	**2003 3**	**2004 2**	**2004 3**
Migration to deme:	**2003**	**3**	*MLE*	**41.34**	**10.60**	**10.79**	**35.05**	**19.43**	*0.16*	N	N
			L	38.15	9.03	9.21	32.13	17.27	*0.14*	N	N
			U	42.71	12.34	12.55	38.16	21.77	*0.17*	N	N
			4Nm	10.3	2.6	2.7	8.8	4.9		N	N
	2004	**2**	*MLE*	**22.40**	5.42	**12.93**	**26.31**	**29.27**	**43.22**	*0.09*	N
			L	19.88	4.23	11.05	23.57	26.33	39.69	*0.08*	N
			U	25.19	6.81	15.02	29.24	32.38	46.96	*0.10*	N
			4Nm	2.1	0.5	1.2	2.4	2.6	3.9		N
	2004	**3**	*MLE*	3.23	**28.04**	5.26	5.28	4.53	6.46	N	*0.41*
			L	2.58	25.98	4.41	4.43	3.75	5.51	N	*0.38*
			U	3.98	30.30	6.22	6.24	5.42	7.51	N	*0.47*
			4Nm	0.6	5.6	1.1	1.1	0.9	1.3	N	



b)

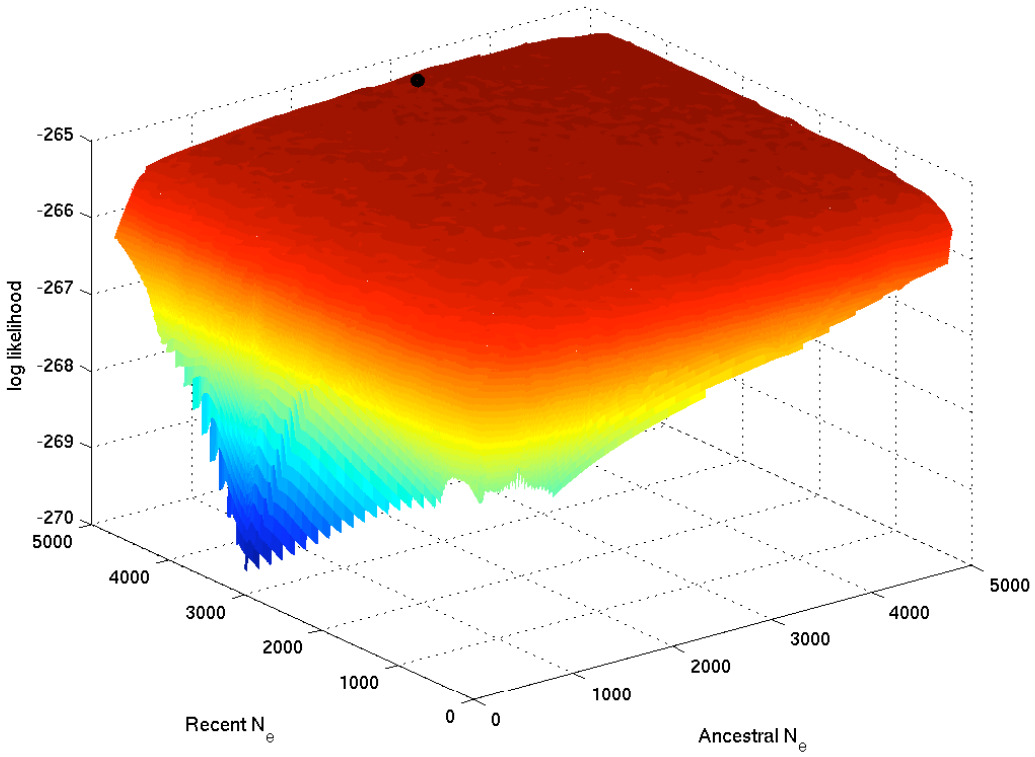


Figure 3: Likelihood surfaces of change in effective population size over the sampling period. The likelihood surfaces were approximated using either a) all the data obtained from 2002-2004, or b) a randomly re-sampled dataset from 2002 of fifty individuals, such that sample sizes in the three sampling periods were equivalent. The maximum likelihood estimate of ancestral and recent Ne is shown as a black point.

we accounted for sample size differences between the sampling years, in particular 2002 versus that of later years, the analysis of change in N_e yielded different results. The likelihood surface of AN_e and RN_e was essentially flat with a MLE of approximately 3115 and 4793 individuals, for AN_e and RN_e, respectively. We furthermore, conducted the analysis on a broader scale of effective population size (10000 individuals on each axis), and the results were essentially the same, a flat surface with no apparent MLE (results not shown). Again 95% confidence limits indicate that randomly sampling the dataset for equivalent sample sizes removes the low signal contained within the data, as evident in the full-data set.

Discussion

Species that exhibit large population fluctuations have been studied intensively from an ecological and population dynamics perspective (see Turchin 2003 for review). However, the analysis of spatial genetic patterns in these species is seldom addressed, with few exceptions (Fuller *et al.* 1997, Vandewoestine *et al.* 1999, Ibrahim *et al.* 2000, Ibrahim 2001, Burton *et al.* 2002, Ehrich *et al.* 2002). The field of phylogeography, which has a principal role of inferring demographic processes from spatial genetic data, has made considerable contributions to understanding demographic processes of focal species (Avise 2000, Hewitt 2001, Knowles & Maddison 2002). Thus, the examination of spatial genetic patterns in species that exhibit complex population cycles should allow for a better understanding of population connectivity and migration rates. However, an examination of studies of spatial genetic pattern in species with complex population cycles indicates that such inferences might be problematic. Spatial genetic analysis of several cyclical species present high levels of inferred migration (Fuller *et al.* 1997, Vandewoestine *et al.* 1999, Burton *et al.* 2002, Ehrich *et al.* 2002). These results are, however, in contradiction to theory where local population size fluctuations are expected to produce greater spatial genetic structure as a result of local genetic drift within local populations (Wright 1940). The effects of extinction and recolonisation on spatial genetic pattern have been evaluated for species with metapopulation structure (Wade & McCauley 1988, Whitlock & McCauley 1990, Ibrahim *et al.* 2000, Ibrahim 2001). The results from these studies in general indicate that the effect of population turnover on genetic differentiation is dependent on the number of individuals colonizing a

deme relative to the number of recurrent migrants between demes (Whitlock & McCauley 1990, Ibrahim 2001). Fewer colonizers produce greater genetic structure due to founder effects, as do fewer migrants. Whether colonizers are sourced from multiple or single demes would also determine the effects of population cycles. Thus, the observed high levels of gene flow in the examples above, and in *Gonometa postica* considered here, might be the result of multiple colonizers, from multiple demes, combined with many recurrent migrants between demes.

The results presented in this manuscript are in agreement with a previous study (Delport 2005 Chapter 4) that inferred high levels of gene flow between sampled localities of *G. postica*. However, given that *G. postica* only survives for 2-9 days (Hartland-Rowe 1992), has no feeding mouthparts (Hartland-Rowe 1992), and may be regarded as an inefficient flier (Delport 2005 Chapter 4, Veldtman 2005), these results are unexpected. The indirect estimates of dispersal (Delport Chapter 4), based on wing load, the ratio of wing area to body mass, could, however, be problematic. High wing loads are characteristic of poor fliers and low wing loads of good fliers (Casey & Joos 1983). However, some species, such as small-bodied geometrid moths, have very low wing loads but are known to be very poor fliers (Rydell *et al.* 1997). Furthermore, some non-feeding saturniid moths, a family closely related to lasiocampids, have been shown to be good dispersers (Waldbauer & Sternberg 1982), despite large body sizes and high wing loads. Delport (2005 Chapter 4) used simulations to determine the effects of population size fluctuations on spatial genetic structure in continuously distributed species. These simulations supported the notion of increased population structure under low levels of gene flow (Wright 1940), yet it could not be shown how population cycles might result in increased spatial genetic homogeneity and thus gross overestimates of gene flow. Thus it still remains to be explicably demonstrated whether the observed spatial genetic patterns in *G. postica* are the result of high dispersal ability or the effects of large inter-annual population size fluctuations.

In this manuscript we determined the levels of gene flow between *G. postica* demes using temporally spaced population genetic data. The results, in general, indicate very high levels of inferred migration between all localities. An anomaly is evident in the greater migration rates between distant, than between adjacent localities (Figure 2). In

contrast, temporal sampling suggested that adjacent localities are likely to seed eruptions in successive years (Figure 3). These migration estimates had narrow confidence limits, however the absence of spatial and temporal patterns in allele frequencies (Figure 1, Table 2) draws question as to which data are contributing to the inference of migrations. Most likely the distribution of rare alleles is contributing to the inference of migration, since the loss of some rare alleles are observed over the sampling period. These reductions in rare allele frequencies can be attributed to either reductions in population size, or insufficient sampling in 2003 and 2004. Delport (2005 Chapter 2) did observe a large decrease in the occurrence of *G. postica* eruptions between 2002 and the subsequent sampling years. Given, however, that the sample sizes in subsequent years were lower it is difficult to determine whether these alleles were actually lost. Similarly, an analysis of the change in effective population size indicated a large reduction in the MLE estimate of effective population size in 2002 versus 2004. Yet, when we corrected for sampling size differences, this result was not apparent. Temporal sampling has been shown to be useful in estimating the change in effective population size of the Mauritius kestrel (Beaumont 2003). However, the distribution of allele frequencies, and whether the population has experienced an eruption or population crash, is likely to affect the success of the method. Given a species with a large skew in the distribution of alleles, and several rare alleles, as in this study, the ability to discriminate population size reduction from insufficient sampling is low, since both would result in the failure to detect rare alleles. However, given several rare alleles and a subsequent population eruption, one would be able to detect whether rare alleles were truly lost or simply not sampled during non-eruptive years. Furthermore, an eruption would allow for rare alleles to become more frequent, and thus allow the tracking of dispersal to neighbouring or distant demes. We believe the inability for temporal genetic data to provide estimates of population size changes and dispersal ability in this manuscript is the result of the decline in eruptions over the sampling period of this study (Delport 2005 Chapter 2). Furthermore, since the demographic estimates in this manuscript are most likely dependent on the frequencies of rare alleles it would be advantageous to increase the number of loci analyzed. This would increase inference power and potentially allow the joint estimation of migration of population size changes using temporally spaced samples.

Acknowledgements

This work was funded by the Mellon Foundation Grant to J. W. H. Ferguson, P. Bloomer and W. Delport, and by a National Research Foundation grant to P. Bloomer (Gun: 2053653). The opinions and views presented in this article are however, not necessarily those of the National Research Foundation. Furthermore, we would like to thank Louis Hauman, Duncan McFadyen and E. O. Oppenheimer & Son for accommodation assistance during sampling. Thanks also to Fourie Joubert and deepthought (http://deepthought.bi.up.ac.za), a 64-node cluster, for computational support.

References

Anderson EC, Williamson EG, Thompson EA (2000) Monte Carlo evaluation of the likelihood for N_e from temporally spaced samples. *Genetics*, **156**, 2109-2118.

Avise JC (2000) Phylogeography: The history and formation of species. Harvard University Press, Harvard.

Beaumont MA (2003) Estimation of population growth or decline in genetically monitored populations. *Genetics*, **164**, 1139-1160.

Beerli P & Felsenstein J. (1999) Maximum-likelihood estimation of migration rates and effective population numbers in two populations using a coalescent approach. *Genetics*, **152**, 763-773.

Beerli P & Felsenstein J. (2001) Maximum-likelihood estimation of a migration matrix and effective population sizes in n subpopulations by using a coalescent approach. *Proceedings of the National Academy of Sciences, USA*, **98**, 4563-4568.

Beerli P (2004) *Migrate Documentation* Version 2.0. School of Computational Science and Department of Biological Sciences, Florida State University, Tallahassee. Florida.

Begon M, Krimbas CB, Loukas M (1980) The genetics of *Drosophila subobscura* populations. XV. Effective size of a natural population estimated by three independent methods. *Heredity*, **45**, 335-350.

Berthier P, Beaumont MA, Cornuet J-M, Luikart G (2002) Likelihood-based estimation of the effective population size using temporal changes in allele frequencies: A genealogical approach. *Genetics*, **160**, 741-751.

Burton C, Krebs CJ, Taylor EB (2002) Population genetic structure of the cyclic snowshoe hare (*Lepus americanus*) in southwestern Yukon, Canada. *Molecular Ecology*, **11**, 1689-1701.

Bjornstad ON, Peltonen M, Liebhold AM & Balstensweiler W (2002) Waves of larch budmoth outbreaks in the European Alps. *Science*, **298**, 1020-1023.

Casey TM, Joos BA (1983) Morphometrics, conductance, thoracic temperature, and flight energetics of Noctuid and Geometrid moths. Physiological Zoology, **56**, 160-173.

Cooper D, Cory JS, Myers JH (2003) Hierarchical spatial structure of genetically variable nucleopolyhedroviruses infecting cyclic populations of western tent caterpillars. *Molecular Ecology*, **12**, 881-890.

Delport W (2005) Temporal and spatial distribution of African Wild Silk Moth, *Gonometa postica*, eruptions in southern Africa. Chapter 2, University of Pretoria thesis, Pretoria, South Africa.

Delport W (2005) Characterisation of six microsatellite loci in the African Wild Silk Moth (*Gonometa postica*, Lasiocampidae). Chapter 3, University of Pretoria thesis, Pretoria, South Africa.

Delport W (2005) The effect of large annual population size fluctuations on spatial genetic pattern in the continuously distributed African Wild Silk Moth (*Gonometa postica*). Chapter 4, University of Pretoria thesis, Pretoria, South Africa.

Drummond AJ, Nicholls GK, Rodrigo AG, Solomon W (2002) Estimating mutation parameters, population history and genealogy simultaneously from temporally spaced sequence data. *Genetics*, **161**, 1307-1320.

Edwards ACF (1972) Likelihood: expanded edition. The Johns Hopkins University Press, London.

Ehrich D, Jorde PE, Krebs CJ, Kenney AJ, Stacy JE, Stenseth NC (2001) Spatial

structure of lemming populations (*Dicrostonyx groenlandicus*) fluctuating in density. *Molecular Ecology*, **10**, 481-495.

Ewens WJ (1979) Mathematical Population Genetics. Springer-Verlag, New York.

Falush D, Kraft C, Taylor NS, Correa P, Fox JG, Achtman M, Suerbaum S (2001) Recombination and mutation during long-term gastric colonization by Helicobacter pylori: Estimates of clock rates, recombination size, and minimal age. *Proceedings of the National Academy of Sciences, USA*, 98, 15056-16061.

Fu Y-X (2001) Estimating mutation rate and generation time from longitudinal samples of DNA sequences. *Molecular Biology and Evolution*, 18, 620-626.

Fuller SJ, Wilson JC & Mather PB (1997) Patterns of differentiation among wild rabbit populations *Oryctolagus cuniculus* L. in arid and semiarid ecosystems of north-eastern Australia. *Molecular Ecology*, **6**, 145-153.

Hartland-Rowe R (1992) The Biology of the wild silkmoth *Gonometa rufobrunnea* Aurivillius (Lasiocampidae) in northeastern Botswana, with some comments on its potential as a source of wild silk. *Botswana Notes and Records*, **24**, 123-133.

Hewitt G (2001) Speciation, hybrid zones and phylogeography – or seeing genes in space and time. *Molecular Ecology*, **10**, 537-549.

Ibrahim KM (2001) Plague dynamics and population genetics of the desert locust: can turnover during recession maintain population genetic structure? *Molecular Ecology*, **10**. 581-591.

Ibrahim KM, Nichols RA, Hewitt GM (1996) Spatial patterns of genetic variation generated by different forms of dispersal during range expansion. *Heredity*, **77**, 282-291.

Jehle RJW, Arntzen WT, Burke AP, Hodl W (2001) The annual number of breeding adults and the effective population size of synoptic newts (*Triturus cristatus*, *T. marmoratus*). *Molecular Ecology*, **10**, 839-850.

Krimbas CB, Tsakas (1971) The genetics of *Dacus oleae*. V. Changes of esterase polymorphism in a natural population following insecticide control: selection of drift? *Evolution*, 25, 454-460.

Knowles LL, Maddison WP (2002) Statistical phylogeography. *Molecular Ecology*, **11**, 2623-2635.

Luikart G, Cornuet J-M, Allendorf FW (1999) Temporal changes in allele frequencies provide estimates of population bottleneck size. *Conservation Biology*, **13**, 523-530.

Nei M, Tajima F (1981) Genetic drift and estimation of effective population size. *Genetics*, **98**, 625-640.

Pollak E (1983) A new method for estimating the effective population size from allele frequency changes. *Genetics*, **104**, 531-548.

Roff DA & Bentzen P (1989) The statistical analysis of mitochondrial DNA polymorphisms: chi-square and the problem of small samples. *Molecular Biology and Evolution*, **6**, 539-545.

Rydell J, Skals N, Surlykke A & Svensson M (1997) Hearing and bat defence in geometrid winter moths. *Proceedings of the Royal Society of London B*, **264**, 83-88.

Turchin (2003) Complex Population Dynamics: a theoretical/empirical synthesis. *Monographs in Population Biology*, **35**. Princeton University Press, Princeton.

Vandewoestijne S, Neve G, Baguette (1999) Spatial and temporal population genetic structure of the butterfly *Aglais urticae* L. (Lepidoptera, Nymphalidae). *Molecular Ecology*, **8**, 1539-1543.

Veldtman R, McGeoch MA & Scholtz CH (2002) Variability in cocoon size in southern African wild silk moths: implications for sustainable harvesting. *African Entomology*, **10**, 127-136.

Veldtman R (2004) The ecology of southern African wild silk moths (*Gonometa* species, Lepidoptera: Lasiocampidae): consequences for their sustainable use. University of Pretoria PhD thesis.

Wade MJ & McCauley DE (1988) Extinction and recolonisation: their effects on the genetic differentiation of local populations. *Evolution*, **42**, 995-1005.

Wang J, Whitlock MC (2003) Estimating effective population size and migration rates from genetic samples over space and time. *Genetics*, 163, 429-446.

Wang J (2001) A pseudo-likelihood method for estimating effective population size from temporally spaced samples. *Genetical Research*, **78**, 243-257.

Wahlund S (1928) Zusammensetzung von Populationen und Korrelation-

serscheinungen von Standpunkt der Verebungslehre aus betrachtet. *Hereditas*, **11**, 65-106.

Waldbauer GP & Sternburg JG. (1982) Long mating flights by Hyalophora cecropia. *J. Lepid. Soc.*, **36**, 154-155.

Waples RS (1989) A generalized approach for estimating effective population size from temporal changes in allele frequency. *Genetics*, **121**, 379-392.

Whitlock MC & McCauley (1990) Some population genetic consequences of colony formation and extinction: genetic correlations with founding groups. *Evolution*, **44**, 1717-1724.

Williamson EG, Slatkin M (1999) Using maximum likelihood to estimate population size from temporal changes in allele frequencies. *Genetics*, **152**, 755-761.

Wright S (1940) Breeding structure of populations in relation to speciation. *American Naturalist*, **74**, 232-248.

Wright S (1943) Isolation by distance. *Genetics*, **28**, 114-138.

Chapter 6

CoalFace: a graphical user interface program for the simulation of coalescence

"I've never had a conflict between teaching and research as some people do because when I'm teaching, I'm doing research"

Raymond Smullyan

CoalFace: a graphical user interface program for the simulation of coalescence

Wayne Delport

Molecular Ecology and Evolution Programme, Department of Genetics, University of Pretoria, Pretoria, 0002, South Africa
wdelport@postino.up.ac.za

Abstract

In this manuscript I describe a computer program that simulates the coalescent process and provides visual outputs of coalescent genealogies to the screen. *CoalFace* is a user-friendly program for teaching the principles of coalescence to both undergraduate and postgraduates in population genetics. In addition, *CoalFace* can generate data for distributions of the time to the most recent common ancestor, number of segregating sites and common diversity indices, from multiple simulations. Windows and Linux (Intel) executables are available at http://www.up.ac.za/academic/genetics/staff/Bloomer/Research/software.htm.

keywords: Coalescence, simulation, genealogies

Accepted for publication in *Molecular Ecology Notes*

The principle of coalescence has gained much recent attention within the population genetics and phylogeography literature recently (Fu & Li 1999, Emerson, Paradis & Thébaud 2001, Rosenberg & Nordborg 2002, Nordborg 2003). This increase is largely attributed to several authors who have worked at developing both coalescent theory (Hudson 1991, Donnelly & Tavaré 1995, Tavaré *et al.* 1997, Bahlo & Griffiths 2000, Wakeley & Aliacar 2001, Excoffier 2004), and analytical software based on the coalescent (Beerli & Felsenstein 2001, Kuhner, Yamato & Felsenstein 1998, Nielsen & Wakeley 2001). Although there is this increase in studies involving the coalescent, few phylogeographic studies take the randomness of the coalescent into account, with notable exceptions (Fleischer *et al.* 1998, Schneider *et al.* 1998, Edwards & Beerli 2000, Irwin 2002). Indeed Knowles (2004) has highlighted the need for coalescent-based modeling in phylogeographic studies. I believe the lack of reference to the coalescent among typical phylogeographical studies, in South Africa at least, is the result of a lack of understanding of how data can be interpreted in light of coalescent models. Furthermore, even though there are several good reviews that explain the difference between gene trees and phylogenetic trees (Smouse 1998, Nichols 2001, Posada & Crandall 2001), and the inappropriateness of the latter in phylogeographical studies, I still find that South African postgraduates and researchers involved in phylogeography misunderstand the implications of coalescent theory for phylogeographical analyses. To this end I have developed a software program, *CoalFace*, which simulates the coalescent process, with the aim of teaching coalescence to both undergraduate students and workshop participants.

CoalFace is a user-friendly software program, written in the Borland Delphi/Kylix programming language, which simulates the coalescent process. The software is written as a teaching aid is therefore, strongly visually orientated, with the ability to draw coalescent genealogies on screen (Figure 1). *CoalFace* can simulate coalescent genealogies, with or without mutations. Mutations are implemented in *CoalFace*, given a mean mutation rate per site per generation. *CoalFace* can simulate mutations under either an infinite or finite sites/alleles model. In the former, the number of segregating sites is simply output as the number of mutations that have occurred on the genealogy. In addition, *CoalFace* is capable of specifying either a finite sites sequence or finite alleles microsatellite mutation model. In the sequence mutation model, mutations can accumulate according to a JC69 (Jukes & Cantor 1969), F81 (Felsenstein 1981), Kimura 2-parameter (Kimura 1980) or HKY85 (Hasegawa,

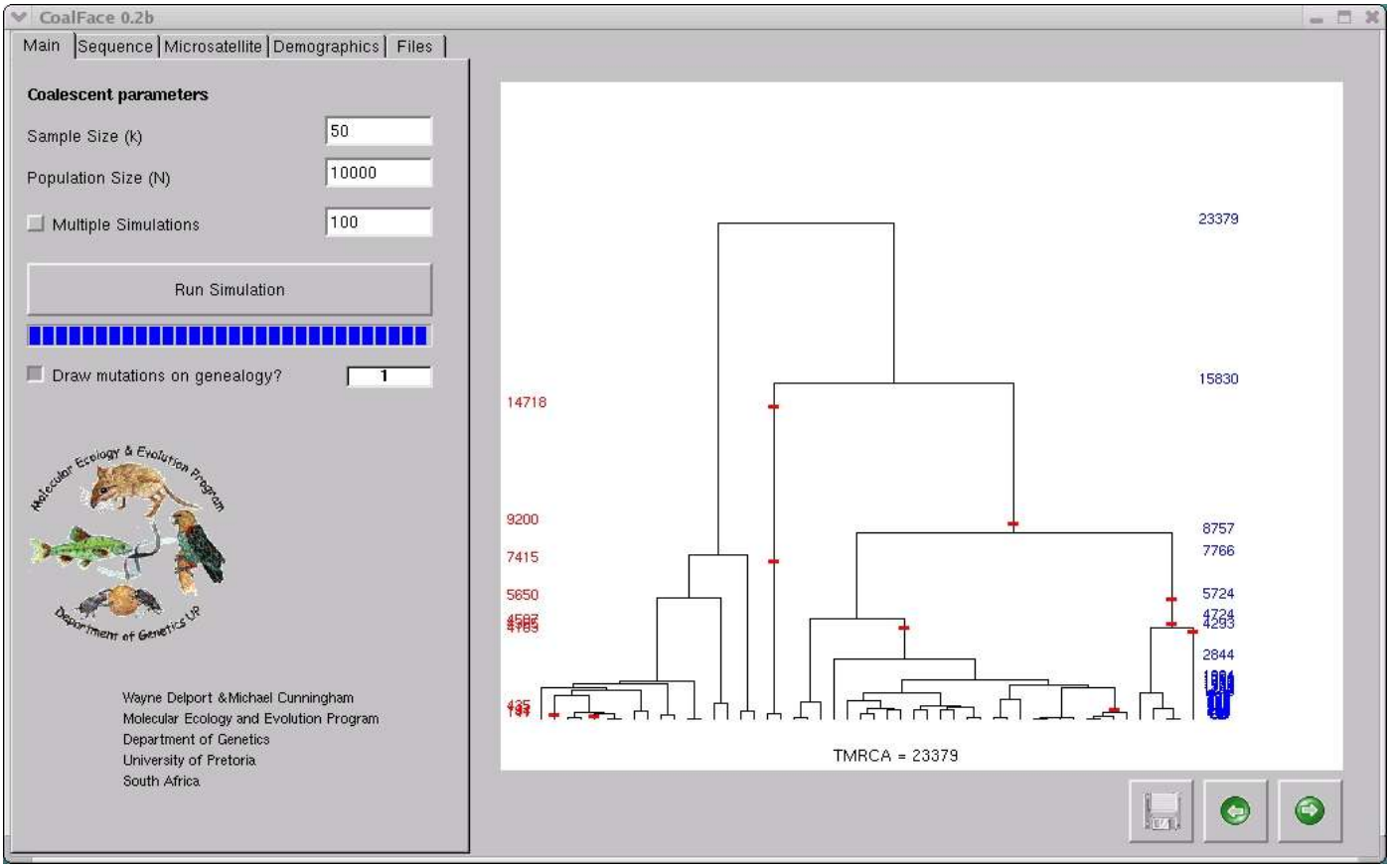


Figure 1: Screenshot of CoalFace showing the visually orientated representation of coalescent genealogies.

Kishino & Yano 1985) substitution model. Among-site rate variation can be specified according to the alpha parameter of a gamma distribution (Yang 1996), as can the proportion of invariable sites. I have found the deviation of the number of segregating sites in an infinite sites model from the number of alleles in a finite sites model especially useful to represent the relationship between mutation rate and the incidence of homoplasy. In addition, the relationship between population size, mutation rate and theta can be investigated, such that students can begin to understand that a large population size with a low mutation rate, is equivalent to a small population, with high mutation rate, in terms of coalescence and population genetic theory. This understanding aids the interpretation of much population genetic literature, which is largely theta-based. In the finite alleles microsatellite model, coalescent simulations are performed independently for each locus, according to a stepwise mutation model (Ohta & Kimura 1973) or a random allele model. Again a comparison of the number of alleles derived from the infinite alleles model, and the finite alleles models, at different mutation rates aids in the understanding of the incidence of homoplasy.

The stochastic nature of the coalescent however, cannot be understood from single simulations of the coalescent. Typically, one should derive distributions of statistics of interest (Figure 2) to gain an understanding of the potential level of stochasticity given a population size and mutation rate. Therefore, I have implemented multiple simulations in *CoalFace*, where the results of each simulation are output to various files. A common statistic of interest is the distribution of times to the most recent common ancestor, and I have used this to teach students the effects of drawing from a given distribution. Furthermore, from an understanding of the distribution of time to the most recent common ancestor, students gain an understanding of why it is notoriously difficult to estimate divergence times from population genetic data. Clearly, the time to the most recent common ancestor is not the only statistic of interest, and *CoalFace* can calculate common diversity indices from sequences generated in the simulation. In additon, I have added a procedure that allows one to output both sequence (nexus, fasta, mrbayes, clustal formats) and microsatellite data to data files. These can then be imported into other software packages for analyses. *CoalFace* can also assemble Arlequin (Schneider *et al.* 2000) files and Arlequin batch files, from multiple simulations. When running multiple simulations, only the last genealogy is drawn, yet the genealogies of previous simulations can be rapidly flipped

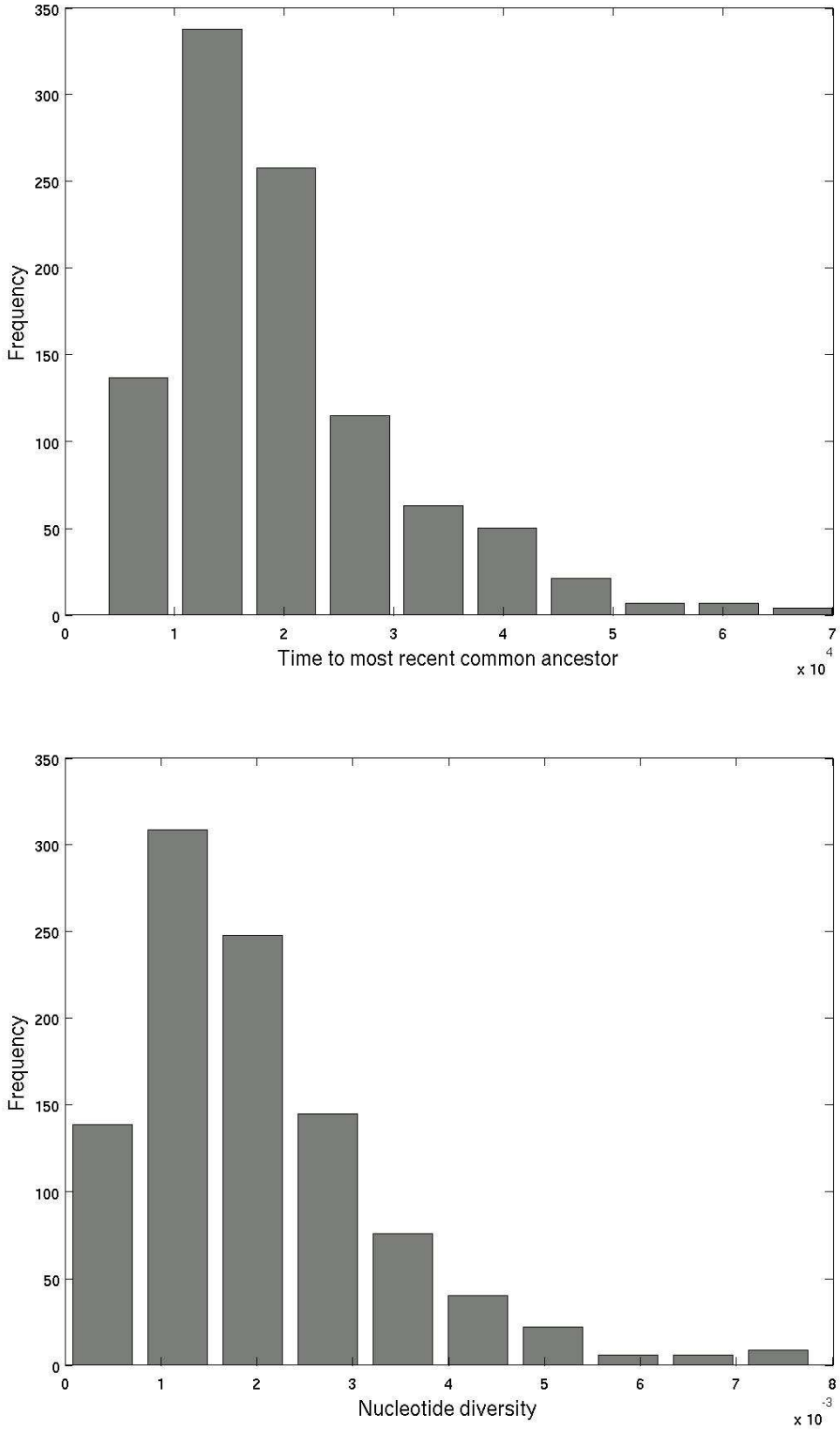


Figure 2: Distributions of (a) time to the most recent common ancestor, and (b) nucleotide diversity derived from 1000 simulations of the coalescent process (k = 50, N = 10000).

over in *CoalFace*. The genealogies are also output to a newick tree file and can be viewed in Rod Page's TreeView (Page 1996).

Finally, I have implemented some basic demographic scenarios in *CoalFace*, such that students can gain an understanding of how increases or decreases in population size, or variance in reproductive success can influence the time to coalescence of a population, and consequently the occurrence of mutations. These demographic simulations are simple and should not be used to test hypotheses of population expansion and contraction; there are far better and faster programs available for these purposes (Excoffier Novembre & Schneider 2000, Grassly, Harvey & Holmes 1999). *CoalFace* is available free from http://www.up.ac.za/academic/genetics/staff/Bloomer/Research/software.htm. Executables for both Windows (NT, 2000, XP) and Linux (intel) are available, as is a simple manual describing how to use the software. Typical run times, on a Linux based Intel 2.0 Ghz Pentium 4 with 384MB Ram, for a single simulation of 50 individuals in a population size of 10000 are on the order of 10-15 seconds. Multiple simulations (1000), with the calculation of diversity indices and exporting of sequence data files take approximately 90 minutes, whereas multiple simulations (1000) with no diversity index calculations take approximately 15-30 minutes. These test runs were performed by sampling 50 individuals from a population size of 10000, and reducing the population size can increase the speed of the runs.

Acknowledgements

This work was partially funded from a Mellon Foundation grant to Wayne Delport and Prof. J. Willem H. Ferguson, and we would like to thank Wayne Delport's PhD supervisors; Prof Paulette Bloomer and Prof J. Willem H. Ferguson, for affording him the time to develop this software program. In addition, we would like to thank participants at the 2003 Phylogeography workshop, presented by the Systematics Society of Southern Africa, for test-driving an earlier version of this software. Finally, we would like to thank Joe Felsenstein for answering some of our coalescence related questions. Also thanks to Michael Cunningham for useful and thoughtful discussion.

References

Bahlo M, Griffiths RC (2000) Inference from gene trees in a subdivided population. *Theoretical Population Biology*, **57**, 79-95.

Beerli P, Felsenstein J (2001) Maximum likelihood estimation of a migration matrix and effective population sizes in n subpopulations by using a coalescent approach. *Proceedings of the National Academy of Sciences, USA*, **98**, 4563-4568.

Donnelly P, Tavaré S (1995) Coalescents and the genealogical structure under neutrality. *Annual Review of Genetics*, **29**, 401-421.

Edwards SV, Beerli P (2000) Gene divergence, population divergence, and the variance in coalescence time in phylogeographic studies. *Evolution*, **54**, 1839-1854.

Emerson BC, Paradis E, Thébaud C (2001) Revealing the demographic histories of species using DNA sequences. *Trends in Ecology and Evolution*, **16**, 707-716.

Excoffier L (2004) Patterns of DNA sequence diversity and genetic structure after a range expansion: lessons from the infinite-island model. *Molecular Ecology*, **13**, 853-864.

Excoffier L, Novembre J & Schneider S (2000) SIMCOAL: a general coalescent program for the simulation of molecular data in interconnected populations with arbritary demography. *Journal of Heredity*, **91**, 506-510.

Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, **17**, 368-376.

Fleischer RC, McIntosh CE, Tarr CL (1998) Evolution on a volcanic conveyor belt: using phylogeographic reconstructions and K-Ar-based ages of the Hawaiian Islands to estimate molecular evolutionary rates. *Molecular Ecology*, **7**, 533-545.

Fu Y-X, Li W-H (1999) Coalesceing into the 21st Century: An overview and prospects of Coalescent theory. *Theoretical Population Biology*, **56**, 1-10.

Grassly NC, Harvey PH & Holmes EC (1999) Population dynamics of HIV-1 inferred from gene sequences. *Genetics*, **151**, 427-438.

Hasegawa M, Kishino H, Yano T (1985) Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, **22**, 160-174.

Hudson R (1991) Gene genealogies and the coalescent process. In Oxford Surveys in Evolutionary Biology vol 7. Futuyma D, Antonovics J (eds). Oxford University Press, Oxford.

Irwin DE (2002) Phylogeographic breaks without geographic barriers to gene flow. *Evolution*, **56**, 2383-2394.

Jukes TH, Cantor C (1969) Evolution of protein molecules. In Mammalian Protein Metabolism. Munro MN (ed). Academic Press, New York.

Kimura M (1980) A simple model for estimating evolutionary rates of base substitution through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, **16**, 111-120.

Knowles LL (2004) The burgeoning field of statistical phylogeography. *Journal of Evolutionary Biology*, **17**, 1-10.

Kuhner MK, Yamato J, Felsenstein J (1998) Maximum likelihood estimation of population growth rates based on the coalescent. *Genetics*, **149**, 429-434.

Nichols R (2001) Gene trees and species trees are not the same. *Trends in Ecology and Evolution*, **16**, 358-364.

Nielsen R, Wakeley J (2001) Distinguishing migration from isolation: a Markov Chain Monte Carlo approach. *Genetics*, **158**, 885-896.

Nordborg M (2003) Coalescent Theory. In Handbook of Statisical Genetics, 2nd edition. Balding DJ, Bishop M, Cannings C (eds). John Wiley and Sons, Ltd.

Ohta T, Kimura M (1973) A model of mutation appropriate to estimate the number of electrophoretically detectable alleles in a finite population. *Genetical Research*, **22**, 201-204.

Page RDM (1996) TREEVIEW: An application to display phylogenetic trees on personal computers. *Computer Applications in the Biosciences*, **12**, 357-358.

Posada D, Crandall KA (2001) Intraspecific gene genealogies: trees grafting into networks. *Trends in Ecology and Evolution*, **16**, 37-45.

Rosenberg NA, Nordborg M (2002) Genealogical trees, coalescent theory and the analysis of genetic polymorphisms. *Nature Reviews Genetics*, **3**, 380-390.

Schneider CJ, Cunningham M, Moritz C (1998) Comparative phylogeography and the history of endemic vertebrates in the wet tropics rainforests of Australia. *Molecular Ecology*, **7**, 487-498.

Schneider S, Roessli D, Excoffier L (2000) Arlequin: a software for population genetics data analysis. Ver 2.000. Genetics and Biometry Lab, Department of Anthropology, University of Geneva.

Smouse PE (1998) To tree or not to tree. *Molecular Ecology*, **7**, 399-412.

Tavaré S, Balding DJ, Griffiths RC, Donnelly P (1997) Inferring coalescence times from DNA sequence data. *Genetics*, **145**, 505-518.

Wakeley J & Aliacar N (2001) Gene genealogies in a metapopulation. *Genetics*, **159**, 893-905.

Yang Z (1996) Among-site rate variation and its impact on phylogenetic analyses. *Trends in Ecology and Evolution*, **11**, 367-372.

Chapter 7

Conclusions

" I love deadlines, I love the wooshing noise they make as they go by."

Douglas Adams

The purpose of the research conducted in this dissertation was to estimate demographic parameters, such as migration and population size changes, using population genetic data in the continuously distributed African Wild Silk Moth, *Gonometa postica*. The African Wild Silk Moth is a species that exhibits large inter-annual population size fluctuations. However, it has proven difficult to estimate these parameters using population genetic data as a result of these fluctuations. Previous theoretical work in this area has shown that the effect of population size fluctuations on spatial genetic structure in metapopulations is dependent on the number of individuals colonizing a deme versus the number of recurrent migrants between demes. In general, however most population size fluctuations generate an increase in spatial genetic structure as a result of local genetic drift. However, in the species considered here high levels of gene flow are inferred from microsatellite data. These results are in agreement with other population genetic studies of cyclical species. In this dissertation I used simulations to determine the dispersal levels at which spatial genetic structure would become panmictic as a result of complex population demographics in a continuously distributed species. Using simulations I could demonstrate the increase in population genetic structure as a result of size fluctuations given low dispersal distances. However, I was unable to show an overestimate of dispersal inferred from population genetic data, as a result of population size fluctuations. Rather, very low dispersal distances, less than 1% of the distribution of the species, resulted in population size fluctuations having no effect on spatial genetic structure versus that of simulations of constant population size. These results seem contradictory to the observed patterns in empirical studies of cyclical species. I believe the failure to show an overestimate in the inference of dispersal from cyclical species is the result of two factors not addressed in this dissertation.

Firstly, I only used Rousset's (2000) estimate of neighbourhood size to monitor the effect of population size fluctuations on spatial genetic pattern. This is the only appropriate statistic for continuously distributed species. However, observations of simulations conducted in this study suggest that neighbourhood size might not be a particularly good statistic to use to monitor the effects of population size fluctuations. Some simulations generated either large over-estimates or large under-estimates of neighbourhood size for some generations. This observation is not the result of an

incorrect implementation of the neighbourhood size calculation, since checking the procedure used in the simulations against calculations performed in Genepop (Raymond & Rousset 1995) using the same data, gave identical results. Leblois *et al.* (2003) have suggested that the incorrect scale of analysis might influence the accurate inference of neighbourhood size. This result was taken into the consideration for the calculation of neighbourhood size in the LatticeFlucIII code, where neighbourhood sizes were averaged over areas of 10σ (where σ is the mean parent-offspring axial dispersal distance) as suggested by Leblois *et al.* (2003). I am currently investigating the influence of outliers in the calculation of neighbourhood size and the suitability of neighbourhood size calculations for unstable populations. Furthermore, I have considered another statistic that may be useful to detect the effects of population size changes on spatial genetic structure in continuous populations. The statistic, A_{DI}, describes the geographic distribution of alleles summed across all loci, where

$$A_{DI} = \sum_{i=1}^{L} \sum_{j=1}^{N} \sum_{k=1}^{n-1} \sum_{l=2}^{n} \frac{D_{kl} S}{D_{kl}}$$

L = number of loci, N = number of alleles for locus i, n = number of individuals/samples, D_{kl} = geographic distance between individual k and l, S = allele sharing coefficient. The value of S is dependent on the degree of sharing for allele j between individual k and l. If both individuals are homozygous for allele j, $S = 1$. If one individual is homozygous for allele j and the other is heterozygous for allele j, $S = 0.75$. If both individuals are heterozygous for allele j then $S = 0.5$. Finally, if only one or neither individual has allele j, $S = 0$. In this way the average geographic distance over which alleles are shared are expressed as a proportion of the total geographic distance over which they could be shared. In a totally homozygous population that is fixed for a single allele, $A_{DI} = 1$, yet in a population where each individual is heterozygous and fixed for unique alleles, $A_{DI} = 0$.

The aforementioned statistic may be suitable for evaluating the effects of population size fluctuations on spatial genetic pattern in continuously distributed species. Preliminary simulation runs under a constant population size and low and high dispersal distances (Figure 1) indicated that the statistic is mostly influenced by allelic

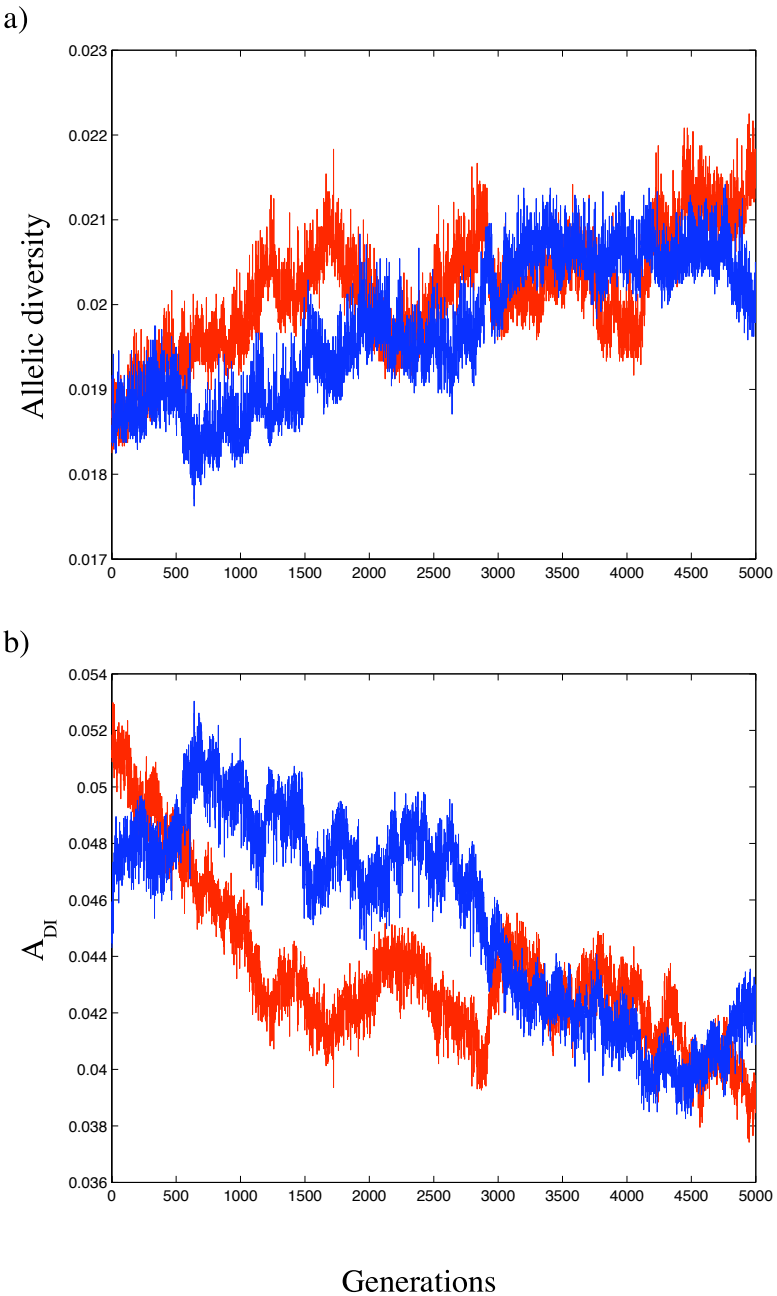


Figure 1: Per generation changes in a) allelic diversity and b) A_{DI} in a population of constant size (N = 125000) over 5000 generations with a microsatellite mutation rate of 2.5 x 10^{-4}. Red = high dispersal, blue = low dispersal.

diversity and not by dispersal distance. An increase in allelic diversity is closely tracked by a decrease in A_{DI}, and dispersal appears to have little effect (Figure 1). Since allelic diversity is typically reduced in populations that experience eruptions this statistic may be of some use. I am currently investigating the properties of this statistic and its utility in tracking the effects of population size fluctuations.

Secondly, the influence of the mutation rate, of the loci analysed, and its effect on inference of spatial genetic patterns from cyclical species has not been considered in this dissertation. The mutation rate of a locus has a direct influence on the observed levels of allelic diversity, and the potential for populations to recover from the effects of genetic drift when populations crash. Furthermore, given low allelic diversity the probability for the same alleles to become fixed as a result of genetic drift in different populations is greater than when allelic diversity is high. The effects of mutation rate of loci will be investigated with future simulations. Furthermore, the number of loci analysed is likely to determine the extent to which one can infer dispersal patterns in cyclical species. Since allelic diversity is generally reduced by population cycles, more loci may be required for such analysis, than in species with stable population dynamics.

From the point of view of understanding the population dynamics of *G. postica* in southern Africa several questions still need to be addressed. Apart from the theoretical considerations of population size fluctuations explored above a thorough understanding of the population biology of *G. postica* is required before recommendations regarding the sustainability of a Wild Silk Industry can be made. Firstly, the interaction between climatic factors and eruptions needs to be explored. In Chapter 2 I showed that eruptions, in general, were correlated with total rainfall. The steady decrease in total rainfall in the Kalahari over the past three years has resulted in substantially lower numbers of *G. postica* eruptions. However, the spatial heterogeneity in eruptions does not appear to be the result of spatial heterogeneity in rainfall, yet this should be explored further with suitable climatic modeling approaches. The interaction between *G. postica* larvae and their host plants is another aspect of the biology of this species that is not well understood. The larval requirements, in terms of foliage quantity and quality, for complete development and

pupation should be evaluated. The integration of this information with host-plant phenology, and the timing of adult moth emergence, is necessary to gain an understanding of the complex exogenous and endogenous factors that contribute to eruptions in this species. The work presented in this dissertation is the initiation of a population dynamics and genetics research programme that has the principle aim of constructing population models for this species. The results presented in this dissertation are paramount in the planning of future research, and in particular the planning of the scale at which to conduct population dynamics research. I look forward to contributing to the development of a long-term population dynamics programme which will attempt to entangle the complex population dynamics of this fascinating moth species.

References

Leblois R, Estoup A & Rousset F (2003) Influence of mutational and sampling factors on the estimation of demographic parameters in a “continous” population under isolation by distance. *Molecular Biology and Evolution*, **20**, 491-502.

Raymond M & Rousset (1995) GENEPOP (version 1.2): population genetics software for exact tests and ecumenicism. *Journal of Heredity*, **86**, 248-249.

Rousset F (2000) Genetic differentiation between individuals. *Journal of Evolutionary Biology*, **13**, 58-62.

Appendix I

A population genetics pedigree perspective on the transmission of *Helicobacter pylori*

"Support bacteria. It's the only culture some people have."

unknown

A population genetics pedigree perspective on the transmission of *Helicobacter pylori*

Wayne Delport[1], Michael Cunningham[1], Brenda Olivier[2], Oliver Preisig[3] & Schalk W van der Merwe[2]

[1]Molecular Ecology and Evolution Programme, Department of Genetics, University of Pretoria, Pretoria, 0002, South Africa

[2]Hepatology/GI-research laboratory, Department of Internal Medicine, University of Pretoria, Pretoria, 0002, South Africa

[3]Inqaba Biotech, Pretoria, South Africa

Corresponding Author:

Dr Schalk W van der Merwe
Department of Internal Medicine and Gastroenterology and GI/Hepatology research laboratory
Lab 2.75, Pathology Building, Basic Medical Sciences Campus
University of Pretoria,
South Africa
Email: svdm@doctors.netcare.co.za
Telephone: +27 12 6640187
Fax: +27 12 6648167

Abstract

The inference of transmission pathways for medicinally important bacteria is crucially important to our understanding of pathogens and the design of efficient pathogen control measures. Here we report an analysis of transmission in *Helicobacter pylori*, a major carcinogen, based on simulations of nucleotide sequences across extended familial pedigrees. Previous epidemiological studies of this organism have been characterized by methodological inadequacies, related to the chosen study population, familial samples sizes, and means of inference. The approach followed here offers improvements in the inference of transmission, including (i) the study of a community with a high prevalence of *H. pylori*, (ii) detailed family pedigrees spanning three generations, (iii) high-resolution analysis of nucleotide sequences, and (iv) a model that incorporates the occurrence of mutation and recombination between the times of infection and subsequent sampling for genetic analysis. Contrary to previous studies, our results demonstrate that the transmission of *H. pylori* is predominantly non-familial, with only a low proportion of mother-to-child transmission events. These results are interesting from both a medical and an evolutionary standpoint. In the former, efficient control measures and beliefs about the sources of *H. pylori* infection should be re-evaluated. Evolutionarily, our results contradict the hypothesis of strict vertical transmission, which has been presented as explanation for the strong correlation between human population history and *H. pylori* diversity. Thus the paradox of persistent phylogenetic structure, despite a permissive mode of transmission and extremely high recombination rates, must be solved elsewhere. Here we consider the potential for recombination events to maintain genetic structure in light of horizontal transmission.

Introduction

Helicobacter pylori is a gram-negative bacterium that colonizes the gastric mucosa (1). This infection is associated with chronic gastritis, peptic ulcer disease, MALT lymphoma and gastric adenocarcinoma, has a major impact on public health (2), and has been classified as a group I carcinogen by the International Agency for Research on Cancer (3). Although much is known of the virulence of *H. pylori* (4), the potential transmission pathways for the bacterium are unresolved (5,6). Potential transmission routes include oral-oral and fecal-oral, both with and without intermediate transmission steps (7). These transmission pathways are supported by a plethora of studies demonstrating the presence of *H. pylori* isolates, either by culturing or PCR techniques, both in the mouth environment (see 5 for a review), and in stool samples (8,9,10). Further clues regarding transmission lie in the fact that infection is more prevalent in developing countries, where it can assume epidemic proportions with up to 80% of the population being infected (11). Such observations suggest that *H. pylori* infection may be associated with low socio-economic status, and overcrowded living conditions, as has been suggested previously (12,13). Transmission studies in developed countries suggest a person-to-person mode of transmission (12). Furthermore, epidemiological and DNA-based studies have suggested that a parent-offspring transmission pathway, mostly mother-to-child, is responsible for transmission within the family and that infection outside the family rarely occurs (12,14,15,16,17).

Several methodological problems are consistent across these studies, and can be summarized as (i) a low incidence of infection within the study population, related to its socio-economic status (16,18), (ii) small sample sizes within families (16,17), and (iii) inappropriate methods of inference for transmission (16,17,18). The first of these methodological problems relates to the observations that poor sanitation, overcrowding and generally low socio-economic status are factors that determine susceptibility to *H. pylori* infection (12,13,19). Given that developed countries have shown a decrease in *H. pylori* incidence without targeted intervention (20,21), the *status quo* of *H. pylori* and its transmission route should be addressed in developing countries where incidence is high, and the bacterium continues to present itself as a serious health concern. Secondly, few studies have compared *H.pylori* genotypes from both parents and several children within individual families (16,17). The observation of transmissions from a mother to each of her children should be the means by which the route of infection is evaluated such that one can distinguish socially mediated transmission from a strictly vertical pathway. Third, transmission studies have typically used presence/absence measures of *H. pylori* infection via ^{13}C Urea breath tests (16,18), presence/absence of antibodies (17,18), restriction enzyme digestion of amplified genes (22) or analyses of peptide sequences (23). Although, the latter two methods are an advance over ^{13}C Urea

breath tests as they allow comparison of *H. pylori* genotypes in parents and children, these methods do not provide the high resolution of nucleotide sequence based studies and are more susceptible to convergent changes. Furthermore, previous studies have not considered the potential for mutation, and/or recombination, to occur between the times of transmission and sampling. Understanding the transmission process is an essential step towards controlling the spread of *H. pylori*. In this manuscript we derive DNA sequence data from extended family pedigrees in a high prevalence community, and use population genetics tools and computer simulations to infer the transmission patterns of *H. pylori*. We show that transmission in this population is not predominantly vertical, and that horizontal transmission from the community plays a major role in the spread of *H. pylori*.

Methods

Sampling and gene sequencing

The study population comprised 105 healthy individuals from a rural, South African, black community (Ogies, Mpumulanga) that have been followed as part of a long-term surveillance program on the epidemiology of *H. pylori* (6,7). This population had many of the risk factors that are associated with a high prevalence of *H. pylori* infection (12, 13). Ethical approval for the current study was obtained from the University of Pretoria and the Hospital Review board of the Unitas hospital. Endoscopy was performed in 90 individuals. DNA was isolated directly from gastric biopsy samples and fragments from three housekeeping genes (UreI, UreC & MutY, Figure 1) were PCR amplified and sequenced. Direct DNA extraction of biopsy samples was preferred over culturing since the latter may result in *in vitro* sequence change, and may also reduce genotypic diversity within a sample. Where multiple genotypes were detected in a single biopsy sample, we cloned PCR products and sequenced a selection of these cloned fragments to identify unique genotypes. Further details of gene sequencing are available in the online supporting material on the PNAS website.

Sequence analysis

Sequences, from the three genes, were analysed in order to identify the predominant route of transmission of *H. pylori.* In summary, (i) neighbour-joining phylograms were used to represent phylogenetic structure of *H. pylori* within the community, (ii) the Structure 2 software (24) was used to align within-community genetic diversity sampled in this study with that found in global *H. pylori* samples, and (iii) statistical comparisons and a custom simulation model were developed to make inferences regarding the most likely path of transmission (Table 1). We simulated transmission of *H. pylori* by drawing from the available sequences in the pedigree and from the community (Figure 2). The source for each transmission event was determined by a random draw

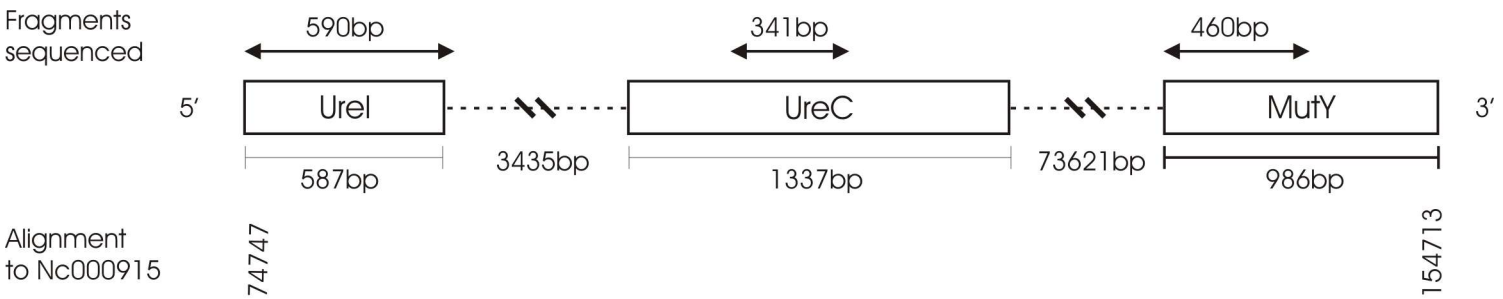


Figure 1: Location of genes sequenced in this study on the *H. pylori* genome. Alignment to the complete *H. pylori* genome sequence on Genbank (NC000915) is shown, as are lengths of fragments sequenced and inter-gene distances in base pairs.

Table 1: Alternative scenarios assessed in the pedigree based simulation model. Each row indicates a transmission scenario, giving rates of infection from different sources. Transmission hypotheses range from predominantly vertical (models 1, 2) to predominantly horizontal transmission, with infection acquired either within families (model 6) or externally, from the wider community (model 7). Further details of the transmission model are available as supporting material on the PNAS website.

	Vertical		**Horizontal**		
	maternal	**paternal**	**sibling**	**household**	**community**
1	0.900	0.050	0.000	0.000	0.050
2	0.475	0.475	0.000	0.000	0.050
3	0.450	0.050	0.000	0.000	0.500
4	0.250	0.250	0.000	0.000	0.500
5	0.250	0.250	0.167	0.167	0.167
6	0.025	0.025	0.500	0.400	0.050
7	0.025	0.025	0.050	0.050	0.850

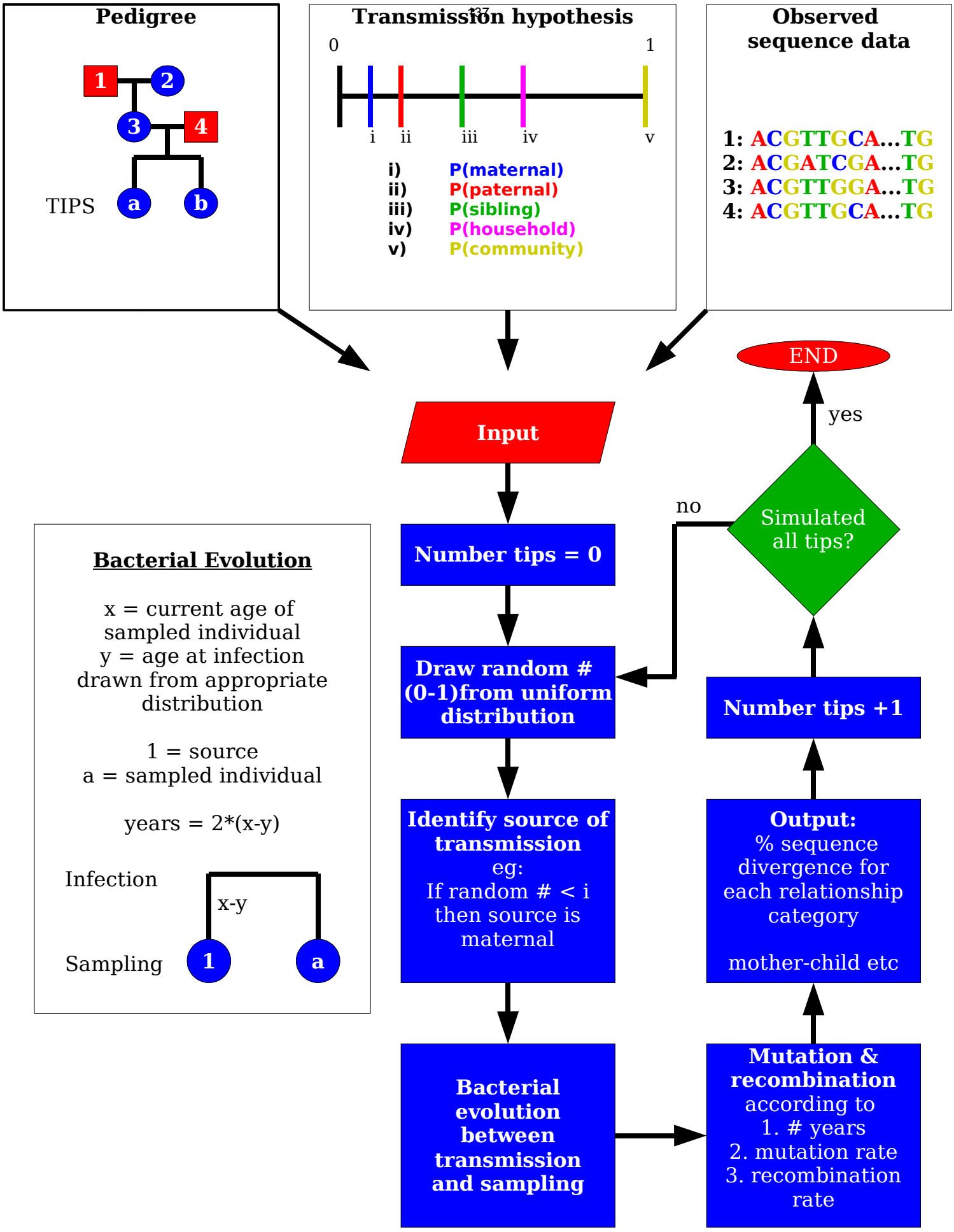


Figure 2: Schematic of the transmission simulation model used to make inferences of the most likely path of transmission of *H. pylori*. Note only a framework of the model is provided. A detailed description is available as online supporting material.

given the transmission hypothesis (Table 1). Simulations were repeated 1000 times for each of these scenarios, to assess general trends in pairwise-sequence comparisons under each transmission hypothesis. The details of these analyses, and a complete description of the simulation model, are published as online supporting information on the PNAS website.

Results

Sequence Analysis

Seventy five individuals (83%) were *H. pylori* positive by PCR and histology. Multiple genotypes were detected in ten (13%) of the individuals sequenced. Co-infection with multiple *H. pylori* genotypes has been suggested in some previous studies (25, 26), although most studies have reported a single genotype per individual. The low level of multiple infection detected in this study, is unlikely to influence the inference of transmission and furthermore, cloned sequences from individuals with multiple genotypes were similar to those from direct sequencing. Consequently, all subsequent analyses were performed assuming one strain per individual. Preliminary sequence analyses show high levels of gene diversity for each of the genes sequenced (Table 2). In general nucleotide diversity is higher for MutY, than for either of the other genes (UreI and UreC). There are low levels of coding substitutions relative to silent substitutions, and thus there is no evidence for selection among the sequenced genotypes at these genes. Given that the fragments sequenced are from general *H. pylori* housekeeping genes this result is expected. Finally, phylogenetic conflict between adjacent segregating sites suggest a minimum estimate of between 23 and 37 recombination events within this sample for each of the three genes.

Global H. pylori diversity and population structure

The purpose of the Structure analysis (see supporting materials and methods) was to place individuals from this study within the global context of previously described *H. pylori* population groups (30). Modern *H. pylori* population groups include hpAfrica1, hpAfrica2, hpEurope and hpEastAsia (30). First, we tested the assignment power of the three genes sequenced in this study, versus eight genes in the former study (30). Two of the three genes sequenced here, MutY and UreI, were common to both studies but UreC was not sequenced in the earlier study. Assignment of global *H. pylori* sequences to the above groups was identical to that in the former study, irrespective of whether UreC was included as missing data, or excluded, and thus provides support for the discriminatory power of our three gene data set. This probably reflects limited recombination between population groups among the fragments sequenced here (Figure 2 in 30), and the exclusion of the largely admixed European samples (see online supporting materials).

Table 2: *H. pylori* diversity indices at three independent loci with mean (and standard deviations) calculated in DnaSp (31). Gene diversity is the probability that two randomly-chosen isolates differ at that locus. Nucleotide diversity is the probability that two selected isolates differ at a randomly chosen nucleotide site. Segregating sites are the number of nucleotide positions that differ among isolates. K_a is the number of coding substitutions per coding site, and K_s is the number of silent substitutions per silent site. Ratios of $K_a/K_s > 1$ indicate selection on the coding sequence of a locus. R_e is the minimum number of recombination events as estimated using the four-gamete test (32) in DnaSP (31).

	UreI	***UreC***	***MutY***
Sequence Length (bp)	590	341	460
Individuals (n)	80	79	79
Gene diversity	0,992 (0,003)	0,998 (0.004)	0,990 (0.004)
Nucleotide diversity	0,041 (0,001)	0,050 (0,002)	0,068 (0,002)
Segregating sites	89	71	105
K_a/K_s	0,045 (0,041)	0,050 (0,060)	0,078 (0.040)
Min. Recombination (R_e)	26	23	37

Secondly we assigned *H. pylori* genotypes from the Ogies community to the previously recognized global population groups. This was performed with a no-admixture model, and with a linkage and admixture model (see online supporting material for details). In the no-admixture model, 47 isolates were assigned to hpAfrica1, 23 to hpAfrica2 and 2 to hpEastAsia. Seventy-two of the seventy-four Ogies individuals were assigned to identical clusters when either two genes (UreI and MutY), or three genes (UreI, UreC, MutY) were used. The incorrect assignments occurred with isolates 112 and 189, which were assigned to hpAfrica1 with three genes, and hpAfrica2 with two genes, for the former, and the converse for the latter. These isolates have contrasting positions in single gene phylograms and their varying assignment to either hpAfrica1 or hpAfrica2 clearly results from recombination events between UreI / UreC and MutY (Figure 3). The linkage and admixture model allows isolates to have mixed origins, and determines the proportion of each individual's nucleotides that are derived from each previously identified ancestral population (30). *H. pylori* nucleotide polymorphisms from the Ogies community were assigned to the ancestral Africa1, ancestral Africa2, ancestral Europe1 and ancestral Europe2 populations (Figure 4). In most cases the proportion of nucleotides derived from a single ancestral population was high. The nucleotide assignments indicate the presence of two clear groups within the community, ancestral Africa1 and ancestral Africa2, and some additional individuals of mixed ancestry, which appear to mostly comprise ancestral Europe2 (Figure 4). Despite the contribution of several population groups to genotype diversity in the Ogies community, within individual ancestral population diversity is lower in this single cohesive population than in the global *H. pylori* sample (Figure 4).

Within-community population structure

Further analyses using the Structure software (24) were conducted to determine the within-community population structure of *H. pylori,* and to estimate recombination parameters. These analyses, using mean likelihoods over five independent runs, estimated the number of subpopulations, K, to be either four or five, as was previously determined for the global *H. pylori* data (30). Some independent replications supported six or seven subpopulations but the inferred likelihoods for these values were not consistently higher (Table 4 published as online supporting material). The admixture model with correlated allele frequencies can tend to overestimate the number of inferred subpopulations as it permits the existence of subpopulations with similar allele frequencies (31). Most nucleotide polymorphisms in the Ogies community are derived from four ancestral populations (Africa1, Africa2, Europe1 and Europe2), with a few admixed individuals, carrying a small proportion of ancestral EastAsia nucleotides.

The Structure software allows the estimation of recombination rate, *r,* and the mean size of recombined sequence chunks. The parameter *r* was calculated, from an admixture model with four

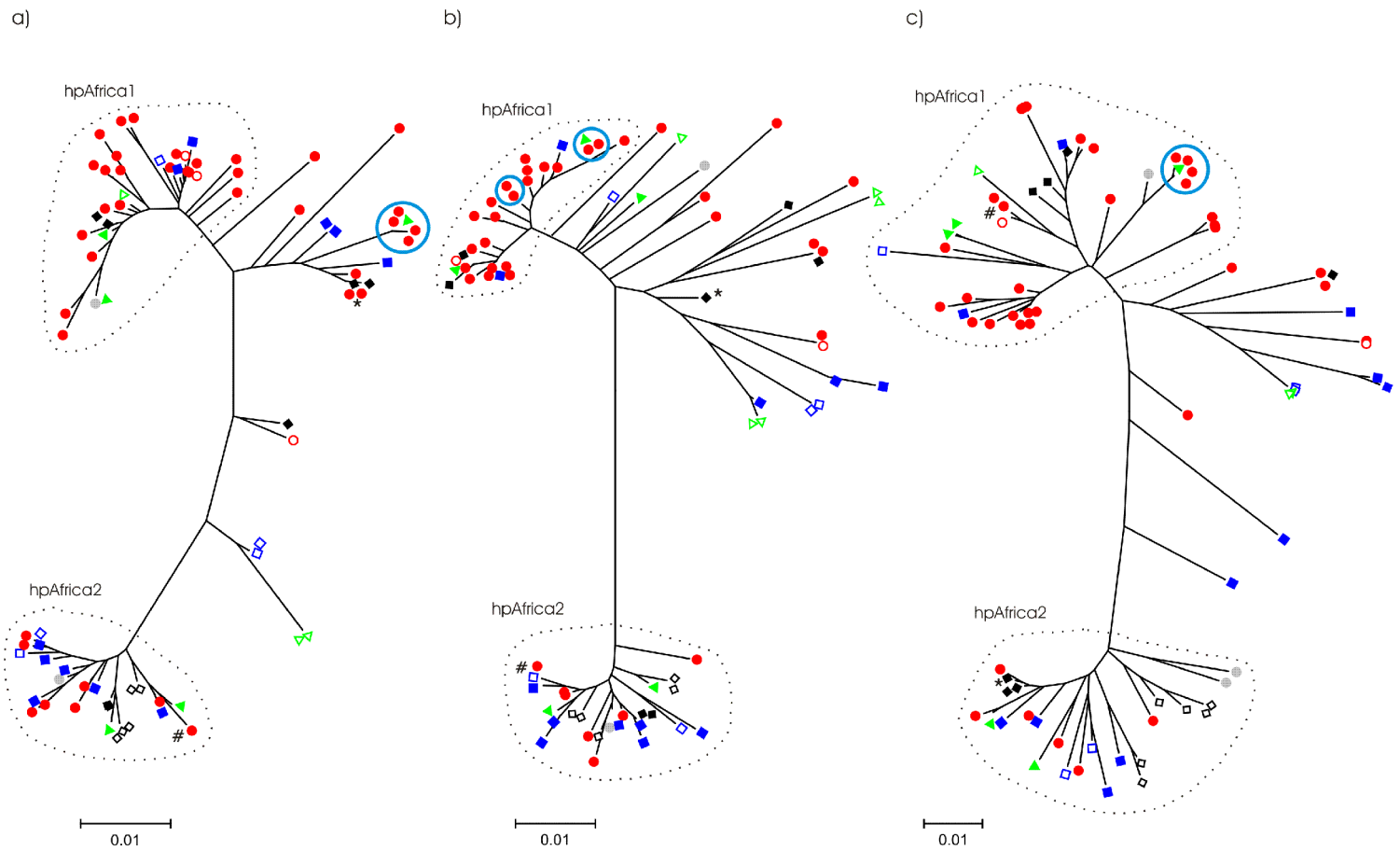


Figure 3: Neighbour-joining phylograms derived from three genes sequenced in this study: a) UreI, b) UreC and c) MutY. The three phylogenetic 'groups' used for the creation of new alleles in the simulation broadly correspond to hpAfrica1, hpAfrica2 and the complement of these. Pedigree information is indicated by symbols, where filled red circles = family 12, filled blue squares = family 13, filled green triangles = family 21, filled black diamonds = family 39, open red circles = family 48, filled gray circles = family 49, open blue squares = family 50, open green triangles = family 51, open black diamonds = family 52. * and # indicate the alternate placements of individuals 112 and 189, respectively.

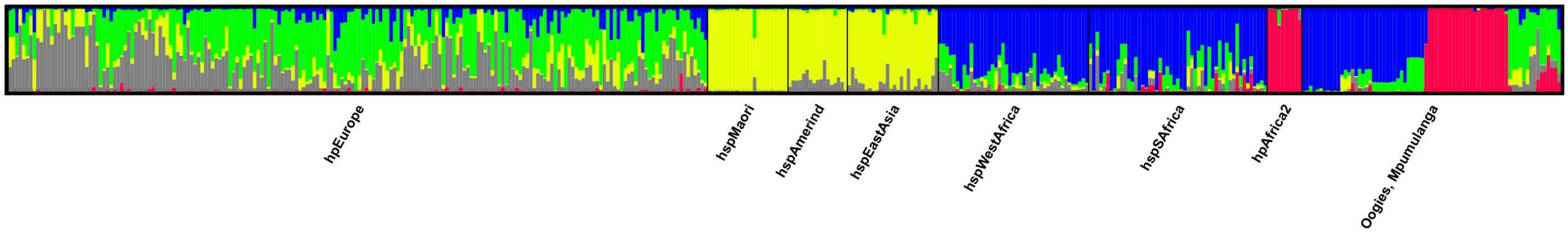


Figure 4: Proportion of nucleotides derived from each of the ancestral populations previously identified (31), for global samples and for within-community samples considered in this study. The proportion of nucleotides was estimated with for three genes, UreI, UreC and MutY, using a linkage and admixture model in Structure 2 (22). UreC was specified as missing data, since this gene was not sequenced in previous studies. Each bar shows the proportion of nucleotides derived from each of five ancestral populations, where colours are as follows: blue: ancestral Africa1, red: ancestral Africa2, gray: ancestral Europe1, green: ancestral Europe2, yellow: ancestral East Asia. The figure was plotted using Distruct (48).

subpopulations, as 0.0007 ± 0.0001 recombination events per nucleotide per year. This estimate is considerably less than the 0.0296 estimated previously (30). This difference implies that either the time since admixture is shorter, or the rate of recombination per nucleotide is lower in our data set. Furthermore, the estimated value of *r* implies an average chunk size of 1428bp versus that of 34bp calculated from the global data set (30). This is consistent with the observation above that within individual levels of admixture are lower in this single community than in the global sample, despite similar levels of genotype diversity.

Transmission analysis

Transmission of *H. pylori* was inferred using two approaches. The first approach used categorical tests to determine whether the number of pairwise comparisons between individuals carrying identical or similar sequences, within a particular relationship category, differed from that expected under a random assignment of genotypes to individuals within the sample. We compared genotypes within mother-child, parent-offspring, sibling, extended family, housemate and spouse relationship classes. To test for associations between genotypes within relationship classes we used Chi-square values, with the test distribution for these pairwise comparisons generated from 1000 random permutations of genotype assignments (Table 3). Results from these Chi-square permutation tests indicate that (i) parents are significantly more likely to share similar *H. pylori* genotypes with their children than are unrelated individuals, (ii) siblings are also likely to share *H. pylori* genotypes, (iii) individuals from the same household show the highest frequency of genotype sharing (evident as large Chi-square values), irrespective of their family relationships, and (iv) spousal partners are no more likely to share *H. pylori* genotypes with each other than with anyone else in the community. These results are consistent with some degree of transmission through childhood social interaction. Most individuals, however, carry substantially different *H. pylori* genotypes, irrespective of their relationships, which suggests that most *H. pylori* infections are acquired outside the family.

The second approach involved the construction of a probabilistic model that used pedigree information and sequence data derived from the study population to simulate transmission processes (Table 1). This simulation model is preferred over categorical tests as it incorporates mutations and recombination events that may have occurred since infection, and allows one to investigate patterns of genetic diversity occurring under multiple contrasting transmission pathways. As with all such models, however, it was first necessary to evaluate simulation results in terms of whether these were comparable with the sample, and whether these are sufficient to discriminate alternate transmission pathways. Since this model simulated a single transmission event for each of 26 individuals one would not expect summary statistics (gene diversity and nucleotide diversity) or phylogenetic structure of the simulated data to differ substantially under the

Table 3: Results of Chi-square permutation tests of association between similar genotypes and particular relationship categories. The test distribution was calculated from 1000 random permutations of the data matrix. Significance at the 0.05 level is indicated in bold.

Relationship	**Chi-square**			***P***		
	UreI	***UreC***	***MutY***	***UreI***	***UreC***	***MutY***
Mother-child	6.9	3.7	10.4	**0.008**	**0.037**	**<0.001**
Parent-offspring	11.2	1.2	12.7	**<0.001**	0.208	**0.001**
Siblings	7.8	0.13	26.9	**0.005**	0.565	**<0.001**
Extended family	0.32	69.9	48.3	0.57	**<0.001**	**<0.001**
Housemates	33.1	17.5	96.5	**<0.001**	**<0.001**	**<0.001**
Spouses	0.7	0.7	0.2	0.068	0.37	0.199

alternate transmission hypotheses. Pairwise comparisons of parent-child divergence, however, should show marked differences, according to the frequency of vertical transmission in a particular transmission scenario. Mismatch distributions, distributions of gene diversity and nucleotide diversity and distributions of segregating sites for three contrasting transmission hypotheses (scenarios 1, 5 and 7 from Table 1) were comparable and similar to the observed data (Figure 6 published as online supporting material). Furthermore, permutation tests in PAUP*4b10 (35) using the observed topology (Figure 3) as a constraint, demonstrated that the simulated data was highly consistent with the observed phylogeny for each of these three alternate transmission hypotheses (in all cases $p = 1.0$). These results indicate that the simulation model does not perturb within-community phylogenetic structure (evident in Figure 3). To assess the potential of mother-child sequence divergence to discriminate alternate transmission hypotheses, we calculated confidence limits, using jackknife and bootstrap procedures, on distribution statistics derived from 1000 simulations. These results indicate that this approach is powerful, with narrow confidence limits on expected statistics (Table 5a published as online supporting material). In particular, there is a shift from a strongly right-skewed distribution of mother-child divergences ($g_1>0$), in vertical transmission models to a strongly left-skewed distribution ($g_1<0$), when infection is predominantly acquired horizontally and outside the family (Table 5a). These results confirm that average pairwise sequence divergences between mothers and their children are low under strict vertical transmission models and high under permissive horizontal transmission models.

In order to infer which of the transmission hypotheses were most likely to generate the observed data we conducted Kolmogorov-Smirnov tests comparing observed distributions of pairwise divergence statistics against those simulated under various transmission scenarios. The observed distribution of within-household sequence divergence values for the 26 focal individuals was significantly different from all the simulated transmission scenarios (Table 6 published as online supporting material). In contrast, the sequence divergence values among siblings were consistent with all the transmission models considered. These cases show rigorous and insufficient discrimination respectively. The inability to discriminate between hypotheses for among housemate comparisons may partly be due to the difficulty in distinguishing instances of vertical and horizontal transmission within households, where individuals can obtain identical genotypes both from specific parent-offspring interactions or from less specific social contacts. Father-child divergence distributions showed significant deviations from the observed data in five of the seven hypothesized transmission scenarios, including those with strictly vertical transmission (Table 6). Observed mother-child divergences were significantly different from both vertical transmission scenarios (Table 6), however elements of both parent-offspring and social transmission are evident in the observed data (Figure 5). In general, scenarios with a high probability of vertical

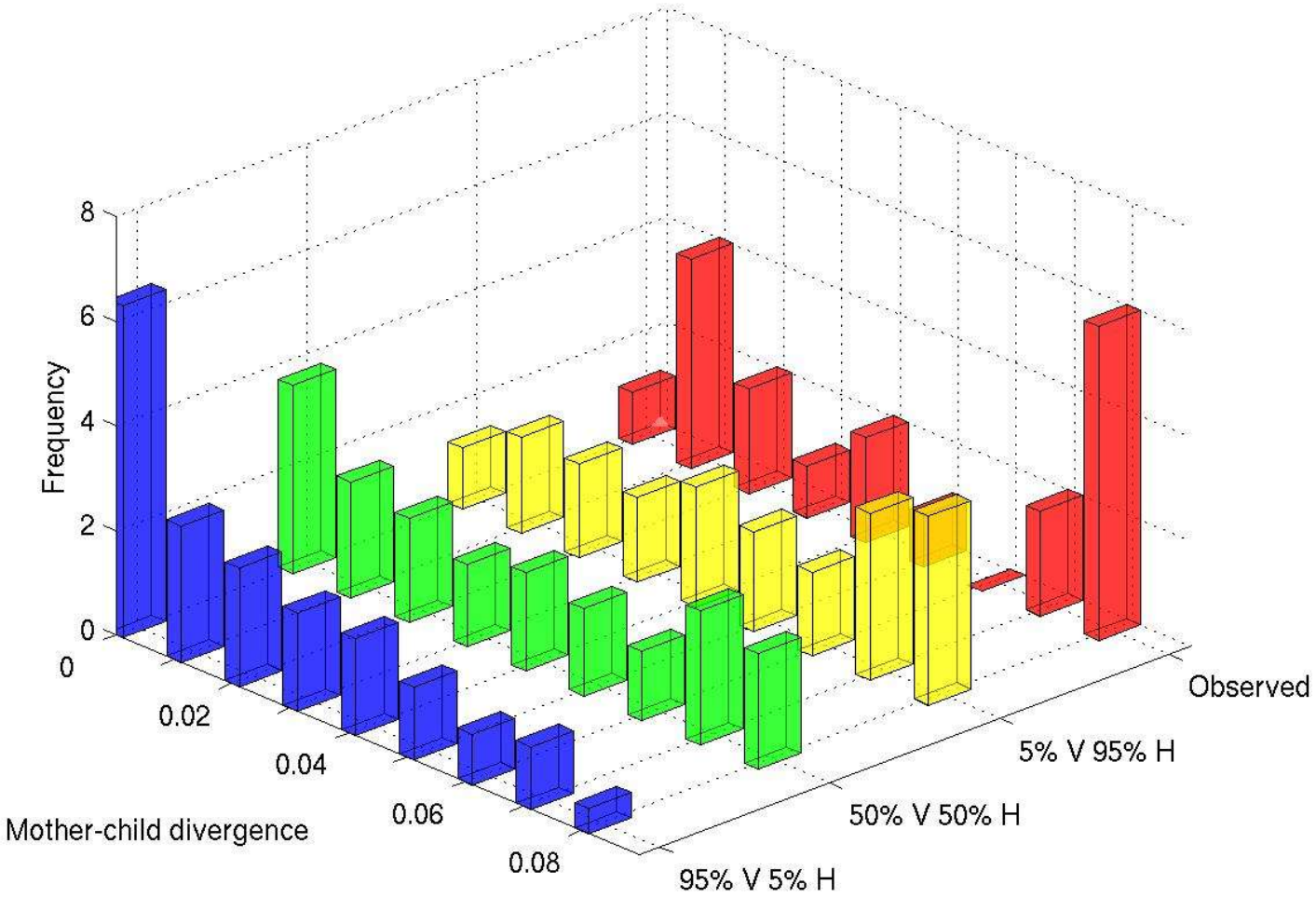


Figure 5: Distribution of mother-child sequence divergence for three simulated transmission hypotheses and the observed data. The mean of 1000 simulations is shown for each of the simulated data sets.

transmission, and scenarios with low probabilities of infection from the community were least consistent with the observed data (Table 6). Jackknife and bootstrap estimates of distribution statistics for the observed data have wider confidence limits than the simulated data (Table 5), as expected given the smaller sample sizes. Nonetheless, the observed mean mother-child sequence divergences are unlikely to have been generated through predominantly vertical transmission. It is difficult to distinguish the observed data from either mixed transmission or predominantly horizontal scenarios, given the wide confidence estimates on mean, median and skewness. Skewness, in particular, has confidence limits that encompass both right-skewed ($g_1>0$) and left-skewed distributions ($g_1<0$), which results from the bimodal distribution of mother-child divergences in the observed data (Figure 5).

Discussion

Previous population genetic studies of *H. pylori* have shown the existence of ancestral population types or strains that are consistent with geographic regions (30,33). These studies indicate the effects that human migrations have had on global genetic diversity within *H. pylori*. Evolutionarily, the existence of multiple strains, each with an ancestral origin that can be geographically determined, provides an independent marker from that of Y-chromosome, microsatellite or mitochondrial DNA studies, for deciphering human history (34). Comparisons of these markers show that DNA sequences from *H. pylori* provide greater resolution, for example, in the separation of Buddhists and Muslim populations in Ladakh, India, than do mtDNA sequences or microsatellites (34). The accurate inference of human migratory patterns using *H. pylori*, however, has been justified on the assumption of strictly vertical transmission (34). To date most studies have suggested predominantly vertical transmission (14,16,18), with most evidence for infection contained within the family unit and a sampling bias towards maternal transmission. We have outlined the methodological problems of these studies and have shown, using simulation modeling, that in a high-prevalence population, transmission of *H. pylori* is not strictly vertical and includes a strong horizontal component derived from the community. Many people in developing countries, and until recently those in the developed countries, live in comparable social conditions and experience similarly high *H. pylori* prevalence to this community (7, 20, 21). Thus the retention of strong ancestral geographic structure within global *H. pylori* sequences (30), within sequences from regional populations (34), and within the single homogeneous community considered in this study requires an alternate explanation. The ancestral population structure observed in Ladakh, is most likely the result of cultural separation of these religious population groups (35,36). The persistence of ancestral *H. pylori* lineages within the essentially homogeneous and intermarried community in the current study, however, suggests that this structure is maintained by bacterial interactions rather

than through separation of human societies alone.

Phylogenetic structuring of global *H. pylori* genotypes probably arose through the isolation of ancestral human populations before the onset of migration and admixture. Subsequent recombination between lineages will disrupt this ancestral population structure but recombination within lineages tends to retain this ancestral structure by homogenizing within group differences. Most likely this retention of ancestral population structure is the result of genome selective mechanisms, which act to limit recombination between different ancestral population groups. Indeed recent microarray experiments show that isolates maintain a genomic framework or scaffold where genes may be shuffled, lost or gained and recombination occurs intra- and extra-genomically, but seldom inter-genomically (37).

From a medical perspective, the inference of predominantly horizontal transmission, from outside the family, affects our epidemiological understanding of *H. pylori* infection, especially in high prevalence communities. The very high levels of gene diversity and the observation that most individuals carry highly divergent genotypes imply the presence of an immense community reservoir of *H. pylori* genotypes, that serves as a source of infection. Many studies have searched for environmental sources of *H. pylori* such as water supplies (13,38,39), food (40), or insect vectors (41,42), but these sources remain poorly substantiated and controversial, perhaps due to the presence of an alternate form of the bacterium that is difficult to culture (5,43). The Ogies community considered here has a reticulated supply of treated tap-water and flushing toilets, hence water and sanitation are unlikely explanations for the high prevalence and diversity of *H. pylori* in this population. An alternative reservoir may be within the community itself, with infection passed from person to person, especially among children. Existing studies have found little evidence for transmission between school children (18) but these results were based on serological analysis in a low prevalence community. Such transmission pathways could account for the strong community-derived transmission component observed in the current study.

Acknowledgments

This work was presented at a plenary session of the Digestive Disease Week conference in May 2004. We would like to thank Mark Achtman, John Atherton and Daniel Falush for pertinent comments on an earlier draft of this manuscript. Wayne Delport would like to thank Prof. P. Bloomer and Prof. J. W. H. Ferguson, for their support. Dr Schalk van der Merwe is a recipient of Astra-Zeneca/ South African Gastroenterology Society Fellowship in Gastroenterology.

References

1. Goodman, K. J. & Correa, P. (1995) *Int. J. Epidemiol.* **24**,875-877.
2. Alm, R. A., Bina, J., Andrews, B. M., *et al.* (2000) *Infect. Immun.* **68**, 4155-4168.
3. NIH Consensus Development Panel. (1994) *J. Am. Med. Assoc.* **272**, 65-9.
4. Covacci, A., Telford, J. L., Del Giudice, G., *et al*. (1999) Science **284**, 1328-33.
5. Dowsett, S. A. & Kowolik, M. J. (2003) *Crit. Rev. Oral Biol. Med.* **14**, 226-233.
6. Goosen, C., Theron, J., Ntasala, M., Maree, F. F., Olckers, A., Botha, S. J., Lastovica, A.J., Van der Merwe, S. W. (2002) *J Clin Microbiology* **40**, 205-209.
7. Olivier, B. J., Bond, R. P., Van Zyl, W. P., Delport, M., Slavik, T., Ziady, C., Terhaar, J., Veldboer, C., Lips, M., Lastovica, A., Fransen, J. H., Kusters, J. G. Kuipers, E. J., Van der Merwe, S. W. (in press) Helicobacter
8. Thomas, J. E., Gibson, G. R., Darboe, M. K., Dale, A., Weaver, A. T. (1992) *Lancet* **340**, 1194-1195.
9. Kelly, S. M., Pitcher, M. C. L., Farmery, S. M., Gibson, G. R. (1994) *Gastroenterology* **107**, 1671-1674.
10. Parsonnet, J., Shmuely, H., Haggerty, B. S. (1999) *J. Am. Med. Assoc.* **282**, 2240-2245.
11. Alm, R. A., Ling, L. S., Moir, D. T., *et al.* (1999) *Nature* **397,**176-180.
12. Suerbaum, S., Michetti, P. (2002) *New Eng. J. Med.* **347**, 1175-1186.
13. Bunn, J. E. G., MacKay WG, Thomas JE, *et al.* (2002) *Lett. Appl. Microbiol.* **34**, 450-454.
14. Drumm, B., Perez-Perez, C. I., Blaser, M. J., Sherman, P. M. (1990) *New England Journal of Medicine* **322**, 359-363.
15. Bamford, K. B., Bickley, J., Collins, J. S. A. (1993) *Gut* **34**, 1348-1350.
16. Rothenbacher, D., Bode, G., Berg, G. *et al.* (1999) *J. Infect. Dis.* **179**, 398-402.
17. Samir, K. S., Martin, B., Gold, B. D. *et al.* (2004) *Helicobacter* **9**, 59-68.
18. Tindberg, Y., Bengtsson, C., Granath, F. *et al*. (2001) *Gastroenterol.* **121**, 310-316.
19. Everhart, J. E. (2000) *Gastroenterol. Clin. North Am.* **20**, 559-578.
20. Parsonnet, J., Blaser, M. J., Perez-Perez, G. I. *et al.* (1992) *Gastroenterology* **102**, 41-46.
21. Banatvala, N., Mayo, K., Megraud, F. (1993) *J. Infect. Dis.* **168**, 219-221.
22. Wang, J. T., Sheu, J. C., Lin, J. T. *et al.* (1993) *J. Infect. Dis.* **168,** 1544-1548.
23. Prewett, E. J., Bickley, J., Owen, R. J., Pounder, R. E. (1992) *Gastroenterology* **102**, 829-833.??
24. Pritchard, J. K., Stephens, M., Donnelly, P. (2000) Genetics **155**, 945-959.
25. Miehlke, S., Thomas, R., Guiterrez, O., Graham, D. Y., Go, M. F. (1999) *J. Clin. Microbiol.* **37**, 245-247.
26. Luman, W., Zhao, Y., Ng, H. S., Ling, K. L. (2002) *European Journal of Gastroenterology and*

Hepatology **14**, 521-528.

27. Akashi, H., Hayashi, T., Koizuka, H., Shimoyama, T., Tamura, T. (1996) *J. Gastroenterol.* **31** (Suppl ix), 16-23.
28. Rozas, J., Rozas, R. (1999) *Bioinformatics* **15**, 174-175.
29. Hudson, R. R. & Kaplan, N. L. 1985. *Genetics* **111**, 147-164.
30. Falush, D., Wirth, T., Linz, *et al.* (2003) *Science* **299**, 1582-1585.
31. Falush, D., Stephens, M. & Pritchard, J. K. (2003) *Genetics* **164**, 1567-1587.
32. Swofford, D. L. (1999) Phylogenetic Analysis Using Parsimony (PAUP), version 4.0. Illinois Natural History Survey, Champaign.
33. Achtman, M., Azuma, T., Berg, D. E., Ito, Y., Morelli, G., Pan, Z-J., Suerbaum, S., Thompson, S. A., van der Ende, A., van Doorn, L-J. (1999) *Mol. Microbiol*. **32**, 459-470.
34. Wirth, T., Wang, X., Linz, B. *et al.* (2004). *Proc. Natl. Acad. Sci. USA* **101**, 4746-4751.
35. Kaul, S. & Kaul, H. N. (1992) in Ladakh Through the Ages, Towards a New Identity. (Nataraj Books, Springfield, VA), pp 118-141.
36. Srinivas, S. (1998) The Mouths of People, the Voice of God (Oxford Univ. Press, New York).
37. Raymond, J., Thiberge, J., Kalach, N., Bergeret, M., Dauga, C., Labigne, A. (2004) *Helicobacter*, **9**, 492
38. Hultzen, K., Han, S. W., Enroth, *et al.* (1996) *Gastroenterology* **110**, 1031-1035.
39. Klein, P. D., Graham, D. Y., Gaillour, A. *et al.* (1991) *Lancet* **337**, 1503-1506.
40. Hopkins, R. J., Vial, P. A., Ferrecio, C., *et al.* (1993) *J. Infect. Dis.* **168**, 222-226.
41. Grubel, P., Huang, L., Masubuchi, N. *et al.* (1998) *Lancet* **352**, 788-789.
42. Osato, M. S., Ayub, K., Le, H. *et al.* (1998) *J. Clin. Microbiol.* **36**, 2786-2788.
43. Bode, G., Mauch, F., Maltfertheiner, P. (1993) *Epidemiol. Infect.* **111**, 483-490.
44. Rosenberg, N. H. (2004) *Molecular Ecology Notes* **4**, 137-138.

A population genetics pedigree perspective on the transmission of *Helicobacter pylori*

Wayne Delport[1], Michael Cunningham[1], Brenda Olivier[2], Oliver Preisig[3] & Schalk W van der Merwe[2,4]

[1]Molecular Ecology and Evolution Programme, Department of Genetics, University of Pretoria, Pretoria, 0002, South Africa
[2]Hepatology/GI-research laboratory, Department of Internal Medicine, University of Pretoria, Pretoria, 0002, South Africa
[3]Inqaba Biotech, Pretoria, South Africa

Supporting Materials and Methods

Gene sequencing

These housekeeping genes were chosen, using the calculations of recombination end-point frequency from (1) to span both narrow (UreC-Ure1) and moderate (UreC-MutY) genomic distances. This strategy would facilitate detection of recombination events following transmission between sampled individuals, a process that could potentially obscure patterns of vertical inheritance of *H. pylori* infection. The inter gene distances between UreI and UreC may be sufficient to maintain linkage disequilibrium between these fragments, while the greater distance between UreC and MutY would allow more recombination and, potentially, break down of linkage disequilibrium. PCR products were purified by precipitation with 95% Ethanol and 3M NaAc, and sequence reads were determined on an ABI 3100 capillary sequencer, following cycle-sequencing using the BigDye 3.2 termination reaction.

Sequence analysis

Sequences from the three genes, UreI, UreC & MutY with 80, 79 and 79 genotypes respectively, were imported into Sequence Navigator (Applied Biosystems, California), where they were proof-read and subsequently aligned using ClustalX (2). We performed preliminary analyses of sequence diversity in DnaSp (3), and used Mega2 (4) to construct unrooted neighbour-joining phylograms based on uncorrected *p* distances for each of the three genes. The aim of these preliminary analyses was to identify overall trends, and population genetic structure within the dataset derived from the Ogies community. Subsequent model-based analyses aimed (i) to align the population genetic structure of *H. pylori* in this community with the previously observed global structure (5), (ii) to estimate the degree of recombination across the three genes sequenced, and (iii) to test alternate transmission hypotheses using the gene sequence and pedigree data from the Ogies community. We

address these aims using Bayesian clustering assignment methods, Bayesian estimation of sample parameters and a custom-built transmission simulation model, respectively.

Global H. pylori diversity and population structure

Falush et al. (5) utilised a Bayesian approach (using Structure 2 (6)) to assign global samples of *H. pylori* to ancestral populations using eight loci. The authors performed these assignments using either a no admixture model, which assumes that each individual has derived its ancestry from a single population, or an admixture model that incorporates recombination between nucleotides. In the former model four modern population groups were consistently identified as hpAfrica1, hpAfrica2, hpEurope and hpEastAsia, yet the hpEurope group showed considerable mixed ancestry, probably the result of Europe being populated by several independent migrations of unknown genetic origin (5,7,8). The incorporation of admixture into the assignment model, which assigns individual nucleotides to ancestral populations based on their linkage with adjacent nucleotides, consistently identified five ancestral population clusters namely, ancestral Africa1 (AA1), ancestral Africa2 (AA2), ancestral Europe1 (AE1), ancestral Europe2 (AE2) and ancestral EastAsia (AEA).

The purpose of our analysis was to determine the contribution of these modern and ancestral population groups to the sample of *H. pylori* sequences from a single rural African community. To this end we first identified individuals from a previous data set (5) that were consistently assigned to their respective extant population groups given the genes that were sequenced in this study (UreI, UreC & MutY). The hpEastAsia, hpAfrica2 and hpAfrica1 groups show little to no recombination among modern populations for two of the genes sequenced in this study, but the hpEurope population group shows high levels of recombination, with the genome of each isolate comprising a mosaic of small recombinant fragments (5: Figure 2). Therefore, we excluded European individuals from the no admixture model for assignment of genotypes to modern global population groups. This exclusion is supported by the relatively low proportion of nucleotides derived from hpEurope in the black South African samples sequenced previously (5). The no admixture model was implemented in Structure 2 (6), with correlated allele frequencies, a burn-in period of 10000 iterations, and subsequent sampling for 20000 iterations. In order to assign sequences to ancestral populations, and to allow for mixed ancestry, we used an admixture model in Structure2 (6) which allows separate assignment of ancestry to each nucleotide variant. These analyses were performed with the full dataset from Falush *et al.* (5), including European individuals, combined with the data generated in this study. The admixture model was implemented with correlated allele frequencies, for five subpopulations (as determined in 5) and a burn-in period of 10000 iterations, with a subsequent 20000 iterations of sampling.

Within-community population structure

A second aim of these analyses was to test for the existence of population genetic structure within a single community of *H. pylori* infected individuals, and to estimate the rate of recombination and average size of recombinant chunks based on these data. Structure 2 estimates the number of distinct subpopulations given the observed data. This estimate is based on the premise that linkage disequilibrium should be observed between populations, but rarely within populations (6). In order to assign individuals to *K* subpopulations we performed heuristic analyses with *K* in the range from two to seven, and noted the posterior probabilities of the data given the value of *K*. These posteriors are dubious at best (6) but they can still be used as an *ad hoc* guide to decide which models best fit the data (6). These exploratory simulations were each run with a burn-in of 10000 iterations, and sampling for a further 20000 iterations (Table 4). We then proceeded with the analyses of structure within the Ogies study population under a linkage and admixture model (as in the analysis of global structure), using the number of groups which best fit the data under each of these models.

Transmission analyses

To obtain a preliminary view of the information content of the data in terms of analyzing transmission hypotheses across relationship categories we performed Chi-square permutation tests. These tests examined the association between individuals carrying similar sequences and individuals within a particular category of relationship (mother-child, parent-offspring, siblings, family members or housemates). A permutation approach was necessary to assess the significance of the observed Chi-square test statistic as each individual appears in multiple pairwise comparisons, thus these comparisons are not independent of each other. We used a computer program incorporating the Roff & Bentzen algorithm (9) to produce a series of 1000 randomized test matrices for each comparison, with the number of similar sequences and the number of individuals in the particular relationship category (the marginal values) held constant. To assess the significance of an association we determined the proportion of these randomized matrices that showed more extreme Chi-square values than the observed data. Since mutations may have accumulated since the time of transmission, we scored sequences as similar if they differed by five or fewer substitutions. This criterion was based on a gap observed in the distribution of pairwise sequence differences (the mismatch distribution), with few comparisons that differ by between six to ten nucleotide substitutions. The maximum mutation rate of *H. pylori* has been estimated as $2.28e^{-5}$ mutations per site per year (5). Given a total of 1391 sites sequenced in this study, the probability of a single mutation occurring each year is 0.03. Thus the time required for five mutations to occur is on the order of 160 years, and it is highly unlikely that more than five mutations could have accumulated between the times of transmission and sampling.

Categorical tests alone provide an inadequate representation of *H. pylori* transmission patterns as there may be multiple routes of person to person infection. A probabilistic model of infection, with pedigree and associated sequence data as inputs, was constructed to characterize patterns of genetic diversity expected under more complex transmission scenarios. In the model, transmission was simulated using a broken-stick design (Figure 2), where the source of infection for each target individual was determined by a random draw from a uniform distribution, with varying frequency segments assigned to five relationship categories (Table 1). Given the mode of transmission, the source individual and the associated DNA sequence was identified from the pedigree data. Infection from the community was simulated by drawing from the dataset of available individuals (n=74), and creating a new allele at a rate determined by the observed gene diversity, that is, from the expectation that two randomly chosen sequences are different. The number of nucleotide substitutions to be enforced along this new allele was determined by first randomly choosing one of the three identified phylogenetic groups (Figure 3) at a rate determined by their frequency in the observed data, and then choosing a pairwise difference from the mismatch distribution within these groups for each of the three candidate genes. Substitutions along the new allele were applied according to the appropriate mutation model for each gene as determined in PAUP*4b10 (10) using ModelTest 3.4 (11) and AIC model selection. Substitutions were constrained to the observed variable sites, or to new mutant sites arising at a rate determined by the nucleotide diversity, such that the strong phylogenetic structure evident in *H. pylori* (5) would be retained in the model. Age of infection was drawn from a gamma distribution (mean = 3, alpha = 0.1), and used to calculate the sequence divergence from time of infection to time of sampling. Time since divergence determined the number of mutation and recombination events, with rates of $6.9e^{-5}$ and $4.1e^{-5}$ per nucleotide per annum respectively (1). The probability of mutation and recombination events along the total sequence length of 79440bp spanning the three genes considered in this study was calculated as $1-(1-\alpha)^{79440}$, where α represents the per site per annum mutation or recombination rate. The occurrence of recombination or mutation events was determined by a random draw for both the probability of occurrence and the affected nucleotide along the entire sequence length. Mutation events that occurred within the sequenced gene regions (Ure1, UreC & MutY) were enforced according to the best-fit mutation model for each gene. Similarly, recombination events that occurred within the sequenced gene regions, or that occurred within one recombinant's length of any of these gene regions, drawn from an exponential distribution with a mean of 417bp (1), were enforced along the simulated DNA sequences. Transmission was simulated for 26 individuals, for whom at least one parent's bacterium infection had been sequenced, and was repeated 1000 times for each of seven transmission hypotheses (Table 1). The model only simulated a single transmission event per individual, and thus summary statistics (gene diversity and nucleotide diversity) and phylogenetic structure of the simulated data should not differ substantially under the

alternate transmission hypotheses. However, pairwise-comparisons of parent-child divergence should show marked differences. We performed Archie-Faith-Cranston randomisation tests, with 1000 permutations, in PAUP*4b10 (9), which calculated the probability that the topologies of randomly chosen simulated datasets for each transmission scenario were consistent with that of the observed data topology. To compare results from alternate transmission scenarios we calculated the mean, median and skewness from simulated distributions of mother-child divergences, and used jackknife, with 50% deletion, and bootstrap resampling procedures to calculate confidence limits on these statistics for each of the seven transmission hypotheses. Finally, we used these resampling procedures along with Kolmogorov-Smirnov tests to infer whether any of the seven transmission hypotheses were unlikely to have given rise to the observed data.

References

1. Falush, D., Kraft, C., Taylor, N. S., *et al.* (2001) *Proc. Natl. Acad. Sci. USA* **98**, 15056-16061.
2. Higgins, D. G., Sharp, P. M. (1988) *Gene* **73**, 237-244.
3. Rozas, J., Rozas, R. (1999) *Bioinformatics* **15**, 174-175.
4. Kumar, S., Tamura, K., Nei, M. (2004) *Briefings in Bioinformatics* **5**, 150-163.
5. Falush, D., Wirth, T., Linz, *et al.* (2003) *Science* **299**, 1582-1585.
6. Pritchard, J. K., Stephens, M., Donnelly, P. (2000) *Genetics* **155**, 945-959.
7. Semino, O., Passarino, G., Oefner, P. J., et al. (2000) *Science* **290**, 1155-1159.
8. Chikhi, L., Nichols, R. A., Barbujani, G. & Beaumont, M. A. (2002) *Proc. Natl. Acad. Sci. USA* **99**, 11008.
9. Roff, D. A., Bentzen, P. (1989) *Mol. Biol. Evol.* **6**, 539-545.
10. Swofford, D. L. (1999) Phylogenetic Analysis Using Parsimony (PAUP), version 4.0. Illinois Natural History Survey, Champaign.
11. Posada, D., Crandall, K. A. (1998) *Bioinformatics* **14**, 817-818.

Table 4: Inference of the number of subpopulations within the Ogies community sample. ln P (X|K) is the log likelihood of the data given the specified K. Log likelihoods are given for five separate simulations for each value of K.

K	**ln *P*(X|K)**					**mean ln *P*(X|K)**
	1	2	3	4	5	
2	-4609	-4609	-4613	-4609	-4609	-4610
3	-3923	-4231	-3786	-3780	-3786	-3901
4	-3527	-3570	-3663	-3498	-3599	-3571
5	-3839	-3397	-3255	-3448	-3410	**-3470**
6	-3215	-3042	-5465	-3045	-7091	-4372
7	-3473	-3159	-4233	-3402	-4535	-3760

Table 5: Bootstrap estimates and 95% confidence limits (lower, upper) for distribution statistics from simulated and observed data. Y = mean, M = median, g_1 = skewness

a) Mother-child sequence divergences from simulated data under the seven transmission models.

a)	***Y***	***M***	g_1
1	0.0281(0.0276, 0.0285)	0.0134 (0.0125, 0.0143)	0.7135 (0.6866, 0.7135)
2	0.0315 (0.0311, 0.0320)	0.0183 (0.0172, 0.0193)	0.5238 (0.4984, 0.5492)
3	0.0374 (0.0370, 0.0379)	0.0305 (0.0293, 0.0317)	0.2601 (0.2356, 0.2846)
4	0.0418 (0.0413, 0.0423)	0.0403 (0.0389, 0.0417)	0.0590 (0.0357, 0.0823)
5	0.0361 (0.0356, 0.0366)	0.0251 (0.0240, 0.0261)	0.3230 (0.2981, 0.3478)
6	0.0411 (0.0407, 0.0416)	0.0383 (0.0364, 0.0401)	0.1405 (0.1165, 0.1645)
7	0.0527 (0.0523, 0.0531)	0.0563 (0.0544, 0.0571)	-0.3282 (-0.3503, -0.3061)

b) Distribution statistics of mother-child (n = 19), father-child (n = 16), sibling (n = 34) and within-household (n = 99) sequence divergences from observed data.

b)	***Y***	***M***	g_1
Mother-child	0.0463 (0.0335, 0.0591)	0.0493 (0.0243, 0.0743)	-0.0478 (-0.8343, 0.7385)
Father-child	0.0489 (0.0397, 0.0582)	0.0444 (0.0291, 0.0598)	0.7750 (-0.2901, 1.8402)
Sibling	0.0309 (0.0217, 0.0401)	0.0273 (0.0054, 0.0492)	0.2819 (-0.2793, 0.8430)
Household	0.0406 (0.0364, 0.0449)	0.0418 (0.0345, 0.0491)	0.0879 (-0.2161, 0.3920)

Table 6: Kolmogorov-Smirnov tests comparing observed versus simulated distributions of test statistics under seven transmission scenarios (Table 1). k = Kolmogorov-Smirnov test statistic, p = probability that the observed distribution could be generated by the simulated model. Significance at the 0.05 level is indicated in bold.

	Mother-child		**Father-child**		**Sibling**		**Household**	
	k	p	k	p	k	p	k	p
1	0.395	**0.004**	0.447	**0.002**	0.150	0.643	0.828	**<<0.001**
2	0.346	**0.016**	0.537	**<0.000**	0.113	0.915	0.811	**<<0.001**
3	0.271	0.103	0.294	0.102	0.180	0.407	0.891	**<<0.001**
4	0.216	0.303	0.379	**0.014**	0.164	0.526	0.888	**<<0.001**
5	0.288	0.070	0.436	**0.003**	0.161	0.553	0.790	**<<0.001**
6	0.183	0.508	0.337	**0.040**	0.273	0.053	0.748	**<<0.001**
7	0.153	0.729	0.244	0.259	0.185	0.377	0.925	**<<0.001**

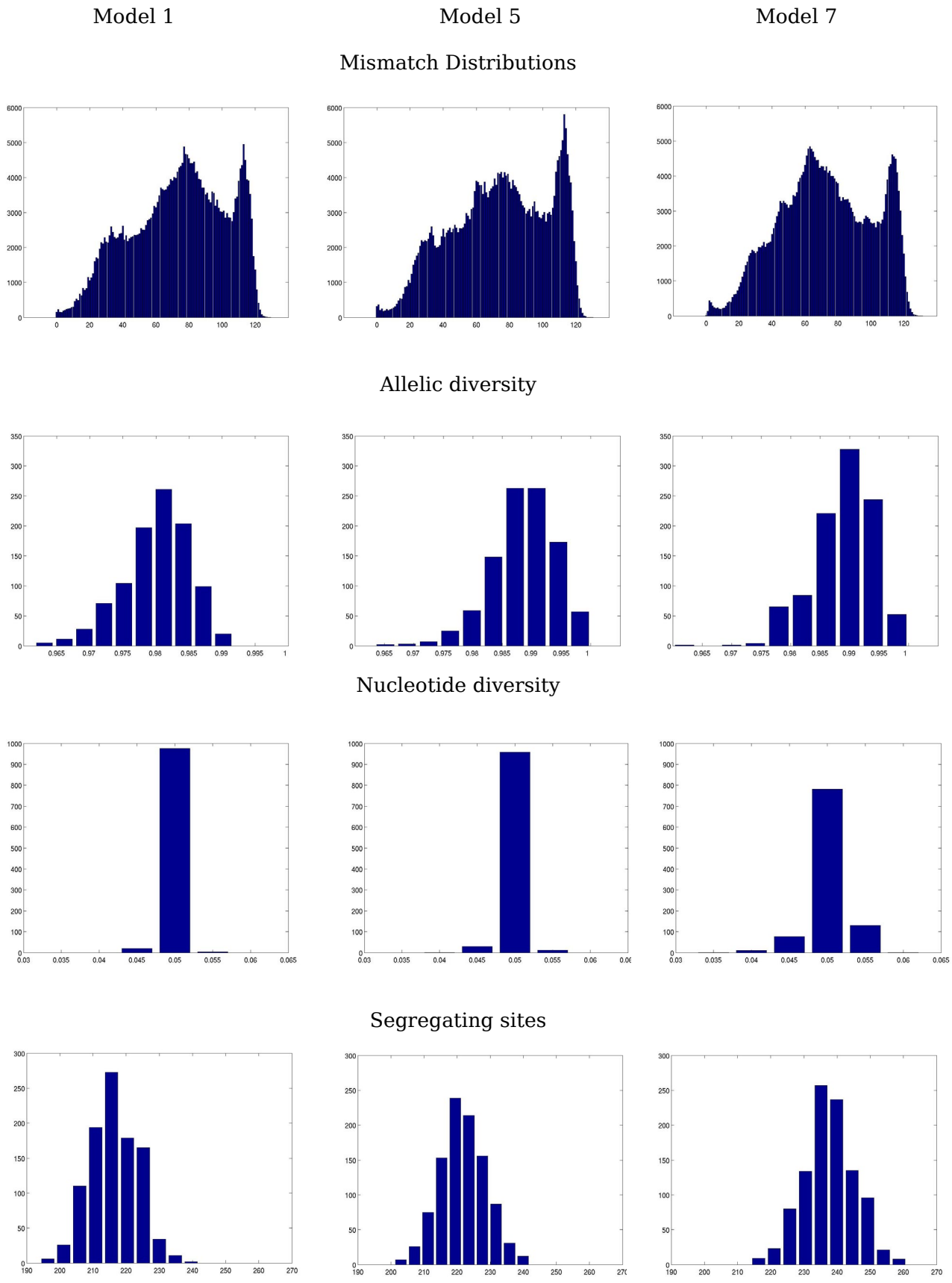


Figure 6: Mismatch distributions, allelic and nucleotide diversity distributions, and distributions of segregating sites under three alternate transmission hypotheses. These results show that the underlying phylogenetic structure of *H. pylori* evident within the sampled community is comparable to that in the simulations.

Appendix II

LatticeFlucIII C Source code

"640K ought to be enough for anybody"

Bill Gates, 1981

```
/*LatticeFlucIII program simulates population turnover in a
continously distributed species for a given number of microsatellite
loci. The program can either be run with unstable of stable
population size, where population size at each generation in the fo
rmer is determined by a Ricker logistic growth model, with a random
draw for growth rate. Rousset's genetic distance between individuals
is calculated at each generation for x subpopulations, each with
population size y. These distances are regressed agai
snt the log of distance to provide an estimate of neighbourhood size.
The purpose of this program is to observe the effect of population
size fluctuations on the estimation of neighbourhood size, using
Rousset's methods.

LatticeFlucIII differs from LatticeFlucII in that it allows the input
of a population size array for population size fluctuations. In this
way the same stochastic simulations can be repeated n times to test
for variance in the observed statistics. Also choice is given for the
dispersal distribution to be leptokurtic or normal.

Rousset (1997) Genetic differentiation and estimation of gene flow
from F-statistics under isolation by distance. Genetics 145: 1219-
1228.
Rousset (2000) Genetic differentiation between individuals. J. Evol.
Biol. 13: 58-62.

Wayne Delport
Molecular Ecology and Evolution Programme
Department of Genetics
University of Pretoria
Pretoria
0002
South Africa
wdelport@postino.up.ac.za

May 2005
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define PI 3.141592654

int SetStartingConditions ( int, int, int, int, float, float );
int AssignSex ( int, int, int );
int PopSizeCalc ( int, float, float, int, int );

float gammaMultiplication ( float );
float gammaRejection ( float );
float gammaTwoPart ( float );
float normal ( float );
float sumofsquares ( float[], float, int );
float mean ( float[], int);

void InitialiseArrays ( int, int, int, int, int, int );
void CreateTempArrays ( int, int, int );
void AssignMuRate ( float, float, int );
void ResetTempArrays ( int, int, int );
void Reproduction ( int, int, int, int, int, int );
void ReproductionII ( int, int, int, int, float, int, int );
void ReproductionIII ( int, int, int, int, float, int, float, int,
```

```
int );
void SamplePopulation ( int, int, int, int, int, int );
void SamplePopulationII ( int, int, int, int );
void WriteDataMatrix ( int, int, int );
void LogOfDistance( float[], int );
void WriteOutfile( int, char[], int );
void PreRegression ( int, int );
void Regression ( float[], float[], float[], int );
void CountAlleles ( int, int );
void CalculateAr ( int, int );
void WritePopMatrix ( int, int, int, int );
void CalculateDiversity ( int, int );
void CalculateAllelicDiversity ( int, int );
void CalculateAlleleDistribution ( int, int, int, int );
void ExportData ( char[], int, int, int );
void ExportSharedAlleleDistances ( char[], int, int, int );
void InitialiseSampleArrays ( int, int );

FILE *ptrOutfile;
FILE *ptrOutfile2;
FILE *ptrInfile;
FILE *ptrStartFile;
FILE *ptrFlucFile;
FILE *ptrDispersalFile;
FILE *ptrOutfile3;

int ****Data;
int ****dataTemp;
int **sexData;
int **sexTemp;
int **Data1, **Data2;
int **Alleles;
int *DataRow, *DataCol;
int *NodeRows;
int *NodeCols;
int *SampledRows;
int *SampledColumns;
int *LocusSSW;
int *LocAlleleNumbers;
int *PopSizeArray;
int *PopFlucArray;
int NodesOccupied;
int reps;
int ActualN;
int NewN;
int popsize;
int WriteDispersalFile;
int WriteSharedAlleleDistances;

float *MutationRate;
float *LogDistance;
float *AdiArray;
float *AdiSDArray;
float *NewLogDistance;
float *ArArray;
float *NewArArray;
float *ArWithinYear;
float *AhatArray;
float *AhatWithinYear;
float *NewAhatArray;
float *DistanceArray;
```

```
float *HeWithinYear;
float *AdiWithinYear;
float *AllelicDivWithinYear;
float *TotalDiv;
float *TotalDivSD;
float *NbSizeAr;
float *NbSizeArSD;
float *NbSizeAhat;
float *NbSizeAhatSD;
float *TotalAllelicDiv;
float *TotalAllelicDivSD;
float *MeanDispersal;
float ***SharedDistances;
float percentOccupied;
float NbSizey1, NbSizey2;
float H;
float AllelicDiv;
float dVariance;
float dVariance2;
float Adi;

char NameString [ 100 ];

int main (int argc, char *argv[ ])
{
  int LatticeRows, LatticeCols;
  int NumLoci;
  int MeanNumAllelesPerLoc;
  int dParameter;
  int dParameter2;
  int RandomRow1, RandomCol1;
  int RandomRow2, RandomCol2;
  int occupied;
  int male;
  int nyears;
  int empty, counter;
  int RandomOffspringX, RandomOffspringY;
  int SampleSize;
  int pCounter;
  int i,j,k,l,m,n,o,p;
  int NodesOccupiedPrev;
  int WriteSampleFiles;
  int DemographicStability;
  int StartingPopulation;
  int Allele1, Allele2, Sex;
  int PopsToSample;
  int PopExtent;
  int PopCrash;
  int DispersalDistrib;
  int CalcMethod;


  float alleleAlpha;
  float MuRateAlpha, MuRateBeta;
  float DirectionY, DirectionX;
  float r;
  float VarEt;
  float meanAr, meanAhat, meanHe, meanAllelicDiv, meanAdi;
  float sdAr, sdAhat, sdHe, sdAllelicDiv, sdAdi;
  float dist, dist2, ar, ahat;
  float testfloat;
```

```
  float Heterozygosity;

  char* DataExport = NULL;
  char* SharedAlleleDistances = NULL;

  if ( argc!=8 )
    printf( "Usage: LatticeFlucIII infile outfile1 outfile2
PopStartFile PopFlucFile DispersalFile SharedAllelesFile\n" );
  else {
    if ( ( ptrInfile = fopen ( argv [ 1 ], "r" ) ) == NULL )
      printf( "Could not open parameter file\n" );
    else {
      if ( ( ptrOutfile = fopen ( argv [ 2 ], "a" ) ) == NULL )
    printf( "Could not open outfile\n" );
      else {
    srand( time( NULL ) );
    DataExport = argv[ 3 ];
    SharedAlleleDistances = argv [ 7 ];
    fscanf ( ptrInfile, "%d%d\n", &LatticeRows, &LatticeCols );
    fscanf ( ptrInfile, "%d\n", &StartingPopulation );
    fscanf ( ptrInfile, "%d\n", &DemographicStability );
    fscanf ( ptrInfile, "%f\n", &percentOccupied );
    fscanf ( ptrInfile, "%d\n", &nyears );
    fscanf ( ptrInfile, "%d\n", &DispersalDistrib );
    if ( DispersalDistrib == 0 ) {
      fscanf ( ptrInfile, "%d\n", &dParameter );
      //printf ( "DispersalDistrib = 0\n" );
    }
    if (DispersalDistrib == 1 ) {
      fscanf ( ptrInfile, "%d%f\n", &dParameter, &dVariance );
      //printf ( "DispersalDistrib = 1\n" );
    }
    if ( DispersalDistrib == 2 ) {
      fscanf ( ptrInfile, "%d%f%d%f\n", &dParameter, &dVariance,
&dParameter2, &dVariance2 );
      //printf ( "DispersalDistrib = 2\n" );
    }
    fscanf ( ptrInfile, "%f\n", &r );
    fscanf ( ptrInfile, "%f\n", &VarEt );
    fscanf ( ptrInfile, "%d\n", &NumLoci );
    fscanf ( ptrInfile, "%f\n", &Heterozygosity );
    fscanf ( ptrInfile, "%d%f", &MeanNumAllelesPerLoc, &alleleAlpha
);
    fscanf ( ptrInfile, "%f%f\n", &MuRateAlpha, &MuRateBeta );
    fscanf ( ptrInfile, "%s\n", NameString );
    fscanf ( ptrInfile, "%d\n", &CalcMethod );
    fscanf ( ptrInfile, "%d\n", &SampleSize );
    fscanf ( ptrInfile, "%d\n", &PopsToSample );
    fscanf ( ptrInfile, "%d\n", &PopExtent );
    fscanf ( ptrInfile, "%d\n", &WriteSampleFiles );
    fscanf ( ptrInfile, "%d\n", &WriteDispersalFile );
    fscanf ( ptrInfile, "%d\n", &WriteSharedAlleleDistances );
    printf ( "Lattice Parameters\n" );
    printf ( "------------------\n" );
    printf ( "LatRows = %d, LatCols = %d\n\n", LatticeRows,
LatticeCols );
    printf ( "Microsatellite parameters\n" );
    printf ( "-------------------------\n" );
    printf ( "Number of Loci = %d\n", NumLoci );
    printf ( "Mean heterozygosity = %f\n", Heterozygosity );
    printf ( "Mean # alleles per locus = %d\n", MeanNumAllelesPerLoc
```

```
);
    printf ( "Allelic diversity alpha parameter = %f\n", alleleAlpha
);
    printf ( "Microsatellite SMM model beta parameters: %f, %f\n\n",
MuRateAlpha, MuRateBeta );
    printf ( "Demographic parameters\n" );
    printf ( "----------------------\n" );
    if ( StartingPopulation == 0 ) {
      printf ( "Started with random population\n" );
      printf ( "Population size is %.0f\n",
LatticeRows*LatticeCols*percentOccupied );
    }
    else
      printf ( "Started with population defined in %s\n", argv [ 4 ]
);
    if ( DemographicStability == 0 )
      printf ( "Constant population size\n" );
    else if ( DemographicStability == 1) {
      printf ( "Population follows Ricker growth model with random
fluctuations in r\n" );
      printf ( "Population growth rate = %f\n", r );
      printf ( "Variance in growth rate = %f\n\n", VarEt );
    }
    else if ( DemographicStability == 2 ) {
      printf ( "Population fluctuations defined in file %s\n", argv [
5 ] );
      if (( ptrFlucFile = fopen ( argv [ 5 ], "r" ) ) == NULL )
        printf( "Could not open FlucFile\n" );
      else {
        PopFlucArray = ( int * ) calloc ( nyears, sizeof( int ) );
        if ( PopFlucArray == NULL )
          printf ( "PopFlucArray memory allocation failure\n" );
        for ( l = 0; l < nyears; l++ ) {
          fscanf( ptrFlucFile, "%d\n", &popsize );
          PopFlucArray [ l ] = popsize;
        }
        fclose( ptrFlucFile );
      }
    }
    printf ( "Total generations = %d\n", nyears );
    if ( DispersalDistrib == 0 ) {
      printf ( "Dispersal drawn from uniform distribution with max =
%d\n", dParameter );
    }
    else if ( DispersalDistrib == 1 ) {
      printf ( "Dispersal drawn from normal distribution with mean =
%d, variance = %f\n", dParameter, dVariance );
    }
    else if ( DispersalDistrib == 2 ) {
      printf ( "Dispersal drawn from leptokurtic distribution with
mean1 = %d, variance1 = %f, mean2 = %d, variance2 = %f\n",
dParameter, dVariance, dParameter2, dVariance2 );
    }
    printf ( "\n" );
    //printf ( "Dispersal = %d\n\n", dParameter );
    printf ( "Calculation of Statistics\n" );
    printf ( "-------------------------\n" );
    if ( CalcMethod == 0 ) {
      printf ( "Sample %d individuals from %d populations each of
lattice size: %d x %d\n", SampleSize, PopsToSample,
PopExtent*dParameter, PopExtent*dParameter );
```

```
    }
    else if ( CalcMethod == 1 ) {
      printf ( "Sample %d individuals from entire lattice for
calculation of statistics, %d repeats for calculation of variance\n",
SampleSize, PopsToSample );
    }
    printf ( "write data file every %d generations\n",
WriteSampleFiles );
    if ( WriteDispersalFile == 1 ) {
      printf ( "Dispersal distances will be written to file %s\n",
argv[ 6 ] );
      printf ( "WARNING: FILES CAN GET VERY LARGE\n" );
    }
    if ( WriteSharedAlleleDistances == 1 ) {
      printf ( "Shared Allele distances will be written to file
%s\n", argv[ 7 ] );
      printf ( "WARNING: FILES CAN GET VERY LARGE\n" );
    }
    InitialiseArrays ( LatticeRows, LatticeCols, SampleSize, NumLoci,
nyears, PopsToSample );
    printf ( "Arrays initialised\n" );
    if ( StartingPopulation == 0 ) {
      SetStartingConditions( NumLoci, LatticeRows, LatticeCols,
MeanNumAllelesPerLoc, alleleAlpha, Heterozygosity );
      AssignSex ( NodesOccupied, LatticeRows, LatticeCols );
    }
    else if ( StartingPopulation == 1 ) {
      if (( ptrStartFile = fopen ( argv [ 4 ], "r" ) ) == NULL )
        printf( "Could not open PopStart File\n" );
      else {
        for ( l = 0; l < LatticeRows; l++ ) {
          for ( m = 0; m < LatticeCols; m++ ) {
        for ( n = 0; n < NumLoci; n++ ) {
          fscanf( ptrStartFile, "%d%d%d\n", &Allele1, &Allele2, &Sex
);
          Data[ l ][ m ][ n ][ 0 ] = Allele1;
          Data[ l ][ m ][ n ][ 1 ] = Allele2;
          //printf( "%d\t%d\t%d\n", Allele1, Allele2, Sex );
          if ( n == 0 )
            sexData[ l ][ m ] = Sex;
        }
          }
        }
        fclose ( ptrStartFile );
      }
    }
    CreateTempArrays ( LatticeRows, LatticeCols, NumLoci );
    AssignMuRate ( MuRateAlpha, MuRateBeta, NumLoci );
    pCounter = ( SampleSize*(SampleSize-1) )/2;
    NodesOccupiedPrev = percentOccupied*(LatticeRows*LatticeCols);
    NodesOccupied = percentOccupied*(LatticeRows*LatticeCols);

    if (( ptrDispersalFile = fopen ( argv[ 6 ], "a" ) ) == NULL )
      printf( "Could not write to dispersal file\n" );
    else {
      PopCrash = 0;
      i = 0;
      while ( ( PopCrash == 0) && ( i < nyears ) ) {
        printf ( "year %d\n", i+1 );
        InitialiseSampleArrays ( PopsToSample, NumLoci );
        ResetTempArrays ( LatticeRows, LatticeCols, NumLoci );
```

```
        if ( DemographicStability == 1 ) { /*1 is for size
fluctuations, 0 for stable population*/
          if ( NodesOccupiedPrev > (LatticeRows*LatticeCols) ) {
        NodesOccupiedPrev = (
percentOccupied*(LatticeRows*LatticeCols) );
          }
          //printf ( "NodesOccupiedPrev is %d\n", NodesOccupiedPrev
);
          NodesOccupied = PopSizeCalc ( NodesOccupiedPrev, r, VarEt,
LatticeRows, LatticeCols );
          //printf ( "Nodes occupied is %d\n", NodesOccupied );
        }
        else if ( DemographicStability == 2 ) {
          NodesOccupied = PopFlucArray [ i ];
          //printf ( "PopFluc from files size is %d\n", NodesOccupied
);
        }
        PopSizeArray [ i ] = NodesOccupied;
        if ( DispersalDistrib == 0 )
          Reproduction ( NodesOccupied, LatticeRows, LatticeCols,
dParameter, NumLoci, i );
        else if ( DispersalDistrib == 1 )
          ReproductionII ( NodesOccupied, LatticeRows, LatticeCols,
dParameter, dVariance, NumLoci, i );
        else if ( DispersalDistrib == 2 )
          ReproductionIII (NodesOccupied, LatticeRows, LatticeCols,
dParameter, dVariance, dParameter2, dVariance2, NumLoci, i );
        WriteDataMatrix ( LatticeRows, LatticeCols, NumLoci );
        for ( o = 0; o < PopsToSample; o++ ) {
          //printf( "PopsToSample is %d\n", PopsToSample );
          if ( CalcMethod == 1 ) {
        //printf ( "SamplePopulationII running\n" );
        SamplePopulationII ( SampleSize, LatticeRows, LatticeCols,
NumLoci ); //entire lattice is sampled
          }
          else if ( CalcMethod == 0 ) {
        SamplePopulation ( SampleSize, LatticeRows, LatticeCols,
NumLoci, PopExtent, dParameter ); //limited geographical extent
          }
          CountAlleles ( ActualN, NumLoci );
          CalculateAllelicDiversity ( ActualN, NumLoci );
          AllelicDivWithinYear [ o ] = AllelicDiv;
          CalculateAr ( ActualN, NumLoci );
          LogOfDistance ( DistanceArray, (ActualN*(ActualN-1))/2 );
          PreRegression ( (ActualN*(ActualN-1))/2, dParameter );
          Regression ( NewLogDistance, NewArArray, NewAhatArray, NewN
);
          ArWithinYear [ o ] = NbSizey1;
          AhatWithinYear [ o ] = NbSizey2;
          CalculateDiversity ( ActualN, NumLoci );
          HeWithinYear [ o ] = H;
          CalculateAlleleDistribution ( ActualN, NumLoci, o, i );
          AdiWithinYear [ o ] = Adi;
          free ( NewLogDistance );
          free ( LogDistance );
          free ( NewArArray );
          free ( NewAhatArray );
          free ( SampledRows );
          free ( SampledColumns );
          free ( DataRow );
          free ( DataCol );
```

```
          free ( Data1 );
          free ( Data2 );
          free ( DistanceArray );
          free ( ArArray );
          free ( AhatArray );
          free ( LocAlleleNumbers );
          free ( Alleles );
          free ( LocusSSW );
        }
        meanAr = mean ( ArWithinYear, PopsToSample );
        meanAhat = mean ( AhatWithinYear, PopsToSample );
        meanHe = mean ( HeWithinYear, PopsToSample );
        meanAllelicDiv = mean ( AllelicDivWithinYear, PopsToSample );
        //printf ( "Allelic div within year = %f\n", meanAllelicDiv
);
        sdAr = sqrt( ( sumofsquares ( ArWithinYear, meanAr,
PopsToSample ) )/(PopsToSample - 1) );
        sdAhat = sqrt( ( sumofsquares ( AhatWithinYear, meanAhat,
PopsToSample ) )/(PopsToSample - 1) );
        sdHe = sqrt( ( sumofsquares ( HeWithinYear, meanHe,
PopsToSample ) )/(PopsToSample - 1) );
        sdAllelicDiv = sqrt ( ( sumofsquares ( AllelicDivWithinYear,
meanAllelicDiv, PopsToSample ) )/(PopsToSample-1) );
        meanAdi = mean ( AdiWithinYear, PopsToSample );
        sdAdi = sqrt ( ( sumofsquares ( AdiWithinYear, meanAdi,
PopsToSample ) )/(PopsToSample-1) );
        NbSizeAr [ i ] = meanAr;
        NbSizeArSD [ i ] = sdAr;
        NbSizeAhat [ i ] = meanAhat;
        NbSizeAhatSD [ i ] = sdAhat;
        TotalDiv [ i ] = meanHe;
        TotalDivSD [ i ] = sdHe;
        TotalAllelicDiv [ i ] = meanAllelicDiv;
        TotalAllelicDivSD [ i ] = sdAllelicDiv;
        AdiArray [ i ] = meanAdi;
        AdiSDArray [ i ] = sdAdi;
        NodesOccupiedPrev = NodesOccupied;
        if ( ( ( (i+1) % WriteSampleFiles ) == 0 ) || ( i == nyears-
1) ) {
          WritePopMatrix ( LatticeRows, LatticeCols, NumLoci, i );
        }
        free ( ArWithinYear );
        free ( AhatWithinYear );
        free ( HeWithinYear );
        free ( AllelicDivWithinYear );
        free ( AdiWithinYear );
        if ( NodesOccupied < 10 ) {
          printf ( "Population crash at generation %d\n", i+1 );
          PopCrash = 1;
        }
        i+=1;
      }
      ExportData ( DataExport, nyears, LatticeRows, LatticeCols );
      if ( WriteSharedAlleleDistances == 1 ) {
        ExportSharedAlleleDistances ( SharedAlleleDistances, nyears,
PopsToSample, SampleSize );
      }
      fclose ( ptrInfile );
      fclose ( ptrOutfile );
      fclose ( ptrOutfile2 );
      if ( WriteSharedAlleleDistances == 1 )
```

```
        fclose ( ptrOutfile3 );
      free ( NodeRows );
      free ( NodeCols );
      free ( TotalDiv );
      free ( TotalDivSD );
      free ( TotalAllelicDiv );
      free ( TotalAllelicDivSD );
      free ( PopSizeArray );
      free ( NbSizeAr );
      free ( NbSizeArSD );
      free ( NbSizeAhat );
      free ( NbSizeAhatSD );
      free ( AdiArray );
      free ( AdiSDArray );
      free ( MeanDispersal );
      if ( WriteSharedAlleleDistances == 1)
        free ( SharedDistances );
      fclose ( ptrDispersalFile );
    }
      }
    }
  }
}

void PreRegression ( int n, int d )
/*According to Rousset 1997 pairwise comparisons of individuals at a
distance less than sigma are not expected to show a linear
relationship. Rousset (1997, 2000) suggests the exclusion of these
comparisons in the calculation of regression.*/
{
  int i;
  int counter;
  int D;

  float lg;
  float ahat, ar;

  NewArArray = ( float * ) calloc ( n, sizeof ( float ) );
  if ( NewArArray == NULL )
    printf ( "NewArArray memory failure\n" );

  NewAhatArray = ( float * ) calloc ( n, sizeof ( float ) );
  if ( NewAhatArray == NULL )
    printf ( "NewAhatArray memory failure\n" );

  NewLogDistance = ( float * ) calloc ( n, sizeof ( float ) );
  if ( NewLogDistance == NULL )
    printf ( "NewLogDistance memory failure\n" );

  counter = 0;
  for ( i = 0; i < n; i++ ) {
    D = DistanceArray [ i ];
    if ( D > d) {
      lg = LogDistance [ counter ];
      NewLogDistance [ counter ] = lg;
      ahat = AhatArray [ counter ];
      NewAhatArray [ counter ] = ahat;
      ar = ArArray [ counter ];
      NewArArray [ counter ] = ar;
      counter+=1;
    }
```

```
  }
  NewN = counter+1;
}

void InitialiseSampleArrays ( int n, int L )
{

  ArWithinYear = ( float * ) calloc ( n, sizeof ( float ) );
  if ( ArWithinYear == NULL )
    printf ( "ArWithinYear memory allocation failure\n" );

  AhatWithinYear = ( float * ) calloc ( n, sizeof ( float ) );
  if ( AhatWithinYear == NULL )
    printf ( "AhatWithinYear memory allocation failure\n" );

  HeWithinYear =  (float * ) calloc ( n, sizeof ( float * ) );
  if ( HeWithinYear == NULL )
    printf ( "HeWithinYear memory allocation failure\n" );

  AllelicDivWithinYear = ( float * ) calloc (n, sizeof ( float ) );
  if ( AllelicDivWithinYear == NULL )
    printf ( "AllelicDivWithinYear memory allocation failure\n" );

  AdiWithinYear = ( float * ) calloc (n, sizeof ( float ) );
  if ( AdiWithinYear == NULL )
    printf ( "AdiWithinYear memory allocation failure\n" );

}

void ExportSharedAlleleDistances ( char a[], int Y, int Pops, int N )
{
  int i,j,k;
  float floatVal;

  if ( ( ptrOutfile3 = fopen ( a, "w" ) ) == NULL )
    printf ( "Could not export Shared Allele Distances\n" );
  else {
    for ( i = 0; i < Y; i++ ) {
      for ( k = 0; k < Pops; k++ ) {
    j = 0;
    floatVal = SharedDistances [ j ][ i ][ k ];
    while ( floatVal != 0 ) {
      if ( j == 0 ) {
        fprintf ( ptrOutfile3, "Gen%dPop%d", i, k );
      }
      fprintf ( ptrOutfile3, ", %f", floatVal );
      j+=1;
      floatVal = SharedDistances [ j ][ i ][ k ];
    }
    fprintf ( ptrOutfile3, "\n" );
      }
    }
  }
}

void ExportData ( char a[], int Y, int Rows, int Cols )
{
  int i;
  int pSize;
  float Tdiv, TdivSD, Ldiv, LdivSD, Nb1, Nb2, D, d;
  float Nb1SD, Nb2SD, ADiv, ADivSD, ADI, ADISD;
```

```
  if ( ( ptrOutfile2 = fopen ( a, "w" ) ) == NULL )
    printf ( "Could not export data\n" );
  else {
    fprintf ( ptrOutfile2, "Gen_%s, PopSize_%s, He_%s, SDHe_%s,
AllelicDiv_%s, SDAllelicDiv_%s, Adi_%s, AdiSD_%s, dispersal_%s,
Density_%s, NBSizeAr_%s, NBSizeArSD_%s, NBSizeAhat_%s,
NBSizeAhatSD_%s\n", NameString, NameString, NameString, NameString,
NameString, NameString, NameString, NameString, NameString,
NameString, NameString, NameString, NameString, NameString );
    for ( i = 0; i < Y; i++ ) {
      pSize = PopSizeArray [ i ];
      D = (pSize/((double)(Rows*Cols)));
      Tdiv = TotalDiv [ i ];
      TdivSD = TotalDivSD [ i ];
      ADiv = TotalAllelicDiv [ i ];
      ADivSD = TotalAllelicDivSD [ i ];
      Nb1 = NbSizeAr [ i ];
      Nb1SD = NbSizeArSD [ i ];
      Nb2 = NbSizeAhat [ i ];
      Nb2SD = NbSizeAhatSD [ i ];
      ADI = AdiArray [ i ];
      ADISD = AdiSDArray [ i ];
      d = MeanDispersal [ i ];
      fprintf( ptrOutfile2,
"%d,%d,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f\n", i+1, pSize, Tdiv,
TdivSD, ADiv, ADivSD, ADI, ADISD, d, D, Nb1, Nb1SD, Nb2, Nb2SD );
    }
  }
}

void CalculateAlleleDistribution ( int N, int L, int Pop, int Y )
/*Calculates the statistic Adi, the distribution of shared alleles
with geographical distance.*/
{

  int pc;
  int i,j,k,l;
  int row, col;
  int NumAlleles;
  int testAllele;
  int A1, A2, B1, B2;
  int Testcase;
  int ctr;

  float SharedDistance;
  float Distance;
  float TotalDistance;

  //FILE *ptrTestFile;
  //FILE *ptrTestFile2;
  /*
  if ( ( ptrTestFile = fopen ( "test1.csv", "w" ) ) == NULL )
    printf ( "Could not open test1\n" );
  if ( ( ptrTestFile2 = fopen ( "test2.csv", "w" ) ) == NULL )
    printf ( "Could not open test2\n" );
  */
  Adi = 0;
  SharedDistance = 0;
  TotalDistance = 0;
  ctr = 0;
```

```
  //printf ( "Number of Loci is %d\n", L );
  for ( i = 0; i < L; i++ ) {
    NumAlleles = LocAlleleNumbers [ i ];
    for ( j = 0; j < NumAlleles; j++ ) {
      testAllele = Alleles[ j ][ i ];
      pc = 0;
      for ( k = 0; k < N-1; k++ ) {
    A1 = Data1[ k ][ i ];
    A2 = Data2[ k ][ i ];
    for ( l = k+1; l < N; l++ ) {
      //printf ( "ctr is %d\n", ctr );
      Distance = DistanceArray [ pc ];
      //fprintf ( ptrTestFile2, "%f\n", Distance );
      //printf ( "Distance is %f\n", Distance );
      B1 = Data1[ l ][ i ];
      B2 = Data2[ l ][ i ];
      Testcase = 0;
      if ( A1 == testAllele )
        Testcase += 1;
      if ( A2 == testAllele )
        Testcase += 1;
      if ( B1 == testAllele )
        Testcase += 1;
      if ( B2 == testAllele )
        Testcase += 1;
      if ( Testcase == 4 ) { //All four alleles in both individuals
identical and thus homozygotes
        //printf ( "Shared 4\n" );
        SharedDistance = SharedDistance + 2*Distance;
        //fprintf ( ptrTestFile, "%f\n", 2*Distance );
        if ( WriteSharedAlleleDistances == 1 ) {
          SharedDistances[ ctr ][ Y ][ Pop ] = Distance;
          ctr+=1;
        }
      }
      if ( Testcase == 3 ) {
        //printf ( "Shared 3\n" );
        SharedDistance = SharedDistance + 1.5*Distance;
        //fprintf ( ptrTestFile, "%f\n", 1.5*Distance );
        if ( WriteSharedAlleleDistances == 1 ) {
          SharedDistances[ ctr ][ Y ][ Pop ] = Distance;
          ctr+=1;
        }
      }
      if ( Testcase == 2 ) {
        if ( ( A1 == A2 ) || ( B1 == B2 ) ) {
          SharedDistance = SharedDistance;
        }
        else {
          //printf ( "Shared 2\n" );
          SharedDistance = SharedDistance + Distance;
          //fprintf ( ptrTestFile, "%f\n", Distance );
          if ( WriteSharedAlleleDistances == 1 ) {
        SharedDistances[ ctr ][ Y ][ Pop ] = Distance;
        ctr+=1;
          }
        }
      }
      pc+=1;
      TotalDistance = TotalDistance + 2*Distance;
    }
```

```
      }
    }
  }
  Adi = SharedDistance/TotalDistance;
  //printf ( "Adi is %f\n", Adi );
  //fclose ( ptrTestFile );
  //fclose ( ptrTestFile2 );
}

void CalculateDiversity ( int N, int L )
/*This procedure calculates heterozygosity from the sampled
population*/
{

    int i,j;
    int Row, Column;
    int Allele1, Allele2;

    H = 0;
    for ( i = 0; i < N; i++ ) {
    for ( j = 0; j < L; j++ ) {
        Row = SampledRows[i];
        Column = SampledColumns[i];
        Allele1 = Data[Row][Column][j][0];
        Allele2 = Data[Row][Column][j][1];
        if ( Allele1 != Allele2 ) {
          H+=1;
        }
    }
    }
    H = H/(N*L);
}

void CalculateAllelicDiversity ( int N, int L )
/*Calculates allelic diversity from sampled population*/
{
  int i;
  int NumAlleles, TotalAlleles;

  TotalAlleles = 0;
  for ( i = 0; i < L; i++ ) {
    NumAlleles = LocAlleleNumbers [ i ];
    TotalAlleles+=NumAlleles;
  }
  //printf ( "Total Alleles = %d\n", TotalAlleles );
  AllelicDiv = (float)( TotalAlleles/(float)(N*2*L) );
  //printf ( "Allelic Div is %f\n", AllelicDiv );
}


/*Implement a Ricker model with stochastic r to achieve population
cycles*/
int PopSizeCalc ( int Nt, float Ro, float var, int LatRows, int
LatCols )
{

  float Et;
  float R;
  int sizeP;
  int Ntplus1;
  int k;
```

```
  R = 0;
  Et = normal ( sqrt ( var ) );
  //printf ( "Et = %f\n", Et );
  R = Ro+Et;
  //printf ( "R is %f\n", R );
  k = LatRows*LatCols;
  //printf ( "k is %d\n", k );
  Ntplus1 = floor( Nt*exp( R*(1-Nt/k) ) );

  return Ntplus1;

}

void WritePopMatrix ( int LatRows, int LatCols, int L, int Y )
{
  int i, j, k;
  int Allele1, Allele2, Sex;

  fprintf ( ptrOutfile, "Year_%d\n", Y+1 );
  for ( i = 0; i < LatRows; i++ ) {
    for ( j = 0; j < LatCols; j++ ) {
      for ( k = 0; k < L; k++ ) {
    if ( k == 0 )
      Sex = sexData[ i ][ j ];
    Allele1 = Data[ i ][ j ][ k ][ 0 ];
    Allele2 = Data[ i ][ j ][ k ][ 1 ];
    fprintf( ptrOutfile, "%d\t%d\t%d\n", Allele1, Allele2, Sex );
      }
    }
  }
  fprintf( ptrOutfile, "\n" );
}

void Regression ( float X[], float Y1[], float Y2[], int n )
{
    int i;

    float x,y1,y2,xy1,xy2;
    float SumX,SumY1,SumY2;
    float AvgX,AvgY1,AvgY2;
    float SSX,SSY1,SSY2;
    float SSxy1, SSxy2;
    float ry1,ry2;
    float yint1,yint2;

    SumX = 0;
    SumY1 = 0;
    SumY2 = 0;
    x = 0;
    y1 = 0;
    y2 = 0;
    AvgX = 0;
    AvgY1 = 0;
    AvgY2 = 0;
    for ( i = 0; i < n-1; i++ ) {
    x = X[i];
    y1 = Y1[i];
    y2 = Y2[i];
    SumX+=x;
    SumY1+=y1;
```

```
    SumY2+=y2;
    }

    AvgX = SumX/n;
    AvgY1 = SumY1/n;
    AvgY2 = SumY2/n;

    SSX = 0;
    SSY1 = 0;
    SSY2 = 0;
    SSxy1 = 0;
    SSxy2 = 0;
    ry1 = 0;
    ry2 = 0;
    yint1 = 0;
    yint2 = 0;
    NbSizey1 = 0;
    NbSizey2 = 0;
    for ( i = 0; i < n-1; i++ ) {
    x = pow(( X[i] - AvgX ),2);
    y1 = pow(( Y1[i] - AvgY1 ),2);
    y2 = pow(( Y2[i] - AvgY2 ),2);
    SSX+=x;
    SSY1+=y1;
    SSY2+=y2;
    xy1 = (X[i]-AvgX)*(Y1[i]-AvgY1);
    xy2 = (X[i]-AvgX)*(Y2[i]-AvgY2);
    SSxy1+=xy1;
    SSxy2+=xy2;
    }
    ry1 = SSxy1/SSX;
    ry2 = SSxy2/SSX;
    yint1 = AvgY1 - ry1*AvgX;
    yint2 = AvgY2 - ry2*AvgX;
    NbSizey1 = 1/ry1;
    NbSizey2 = 1/ry2;
    //printf ( "NbSizey1 = %f; NbSizey2 = %f\n", NbSizey1, NbSizey2
);
}

void LogOfDistance( float Array[], int n )
{
    int i;

    float dist, logdist;

    LogDistance = ( float * ) calloc ( n, sizeof( float ) );
    if ( LogDistance == NULL ) {
    printf ( "LofDistance memory allocation failure\n" );
    }
    else {
    for ( i = 0; i < n-1; i++ ) {
        dist = Array[ i ];
        if ( dist == 0 ) {
        dist = 1;
        }
        logdist = log(dist);
        LogDistance[ i ] = logdist;

    }
    }
```

```
}

void CalculateAr ( int N, int L )
{
    int pairwiseCounter;
    int a,b,d,e,q,r;
    int NumAlleles;
    int counter;
    int TestAllele1;
    int Allele1, Allele2, Allele11, Allele12, Allele21, Allele22;
    int Row, Column;
    int Row1, Column1;
    int XSquare, YSquare;

    float X11, X12, X21, X22;
    float AvgXiu, AvgXiub, AvgXu;
    float Term1, Term2;
    float Denominator,Numerator, Denominator2, Numerator2;
    float SumDenominator, SumNumerator, SumDenominator2,
SumNumerator2;
    float SSb, SSw, SSw1;
    float Ar, Ahat;
    float Distance;

    DistanceArray = (float *) calloc ( (N*(N-1))/2, ( sizeof( float
)));
    if ( DistanceArray == NULL ) {
      printf( "DistanceArray memory allocation failure\n" );
    }

    ArArray = (float *) calloc ( ( N*(N-1))/2, ( sizeof( float )));
    if ( ArArray == NULL ) {
      printf( "ArArray memory allocation failure\n" );
    }

    AhatArray = (float *) calloc ( ( N*(N-1))/2, ( sizeof( float )));
    if ( AhatArray == NULL ) {
      printf( "AhatArray memory allocation failure\n" );
    }

    LocusSSW = (int *) calloc ( L, ( sizeof(int)));
    if ( LocusSSW == NULL ) {
      printf( "LocusSSW memory failure\n" );
    }

    pairwiseCounter = 0;
    for ( q = 0; q < N-1; q++ ) {  /*Calculate Ar and Ahat for each
pair of individuals sampled*/
    for ( r = q+1; r < N; r++ ) {
        SumDenominator = 0;
        SumNumerator = 0;
        SumDenominator2 = 0;
        SumNumerator2 = 0;
        for ( d = 0; d < L; d++ ) {
        Numerator = 0;
        Denominator = 0;
        Numerator2 = 0;
        Denominator2 = 0;
        NumAlleles = LocAlleleNumbers[d];
        if ( ( q == 0) && ( r == 1) ) { /*Calculate the SSw over all
pairwise comparisons only once for each locus*/
```

```
            SSw1 = 0;
            counter = 0;
            for ( a = 0; a < N-1; a++ ) {
            for ( b = a+1; b < N; b++ ) {
                Allele11 = Data1[a][d]; /*Allele 1 ind 1 of pairwise
comp*/
                Allele12 = Data2[a][d]; /*Allele 2 ind 1 of pairwise
comp*/
                Allele21 = Data1[b][d]; /*Allele 1, ind 2 of pairwise
comp*/
                Allele22 = Data2[b][d]; /*Allele 2, ind 2 of pairwise
comp*/
                for ( e = 0; e < NumAlleles; e++ ) {
                Term1 = 0;
                TestAllele1 = Alleles[e][d];
                if ( TestAllele1 == Allele11 ) {
                    X11 = 1;
                }
                else {
                    X11 = 0;
                }
                //printf( "allele %d X11 = %f\n", e, X11 );
                if ( TestAllele1 == Allele12 ) {
                    X12 = 1;
                }
                else {
                    X12 = 0;
                }
                //printf( "allele %d X12 = %f\n", e, X12 );
                if ( TestAllele1 == Allele21 ) {
                    X21 = 1;
                }
                else {
                    X21 = 0;
                }
                //printf( "allele %d X21 = %f\n", e, X21 );
                if ( TestAllele1 == Allele22 ) {
                    X22 = 1;
                }
                else {
                    X22 = 0;
                }
                //printf( "allele %d X22 = %f\n", e, X22 );
                AvgXiu = (X11+X12)/2;
                //printf( "allele %d AvgXiu = %f\n", e, AvgXiu );
                AvgXiub = (X21+X22)/2;
                //printf( "allele %d AvgXiub = %f\n", e, AvgXiub );
                    Term1 = pow((X11-AvgXiu),2) + pow((X12-AvgXiu),2)
+ pow((X21-AvgXiub),2) + pow((X22-AvgXiub),2);
                    SSw1+=Term1; /*Keep adding over all alleles and
pairwise comparisons*/
                }
                counter+=1;
            }
            }
            LocusSSW[d] = SSw1;
        }
        SSw1 = LocusSSW[d];
        Denominator = 2*SSw1;
        Denominator2 = SSw1;
/*
```

```
        if ( ( q == 0) && ( r == 1) ) {
            printf( "locus %d, SSW is %f\n", d, SSw1 );
        }
*/
        Allele11 = Data1[q][d];
        Allele12 = Data2[q][d];
        Allele21 = Data1[r][d];
        Allele22 = Data2[r][d];
        SSw = 0;
        SSb = 0;
        //printf( "NumAlleles is %d\n", NumAlleles );
        for ( e = 0; e < NumAlleles; e++ ) {
            TestAllele1 = Alleles[e][d];
            Term1 = 0;
            Term2 = 0;
            //printf( "Test allele is %d\n", TestAllele1 );
            if ( TestAllele1 == Allele11 ) {
            X11 = 1;
            }
            else {
            X11 = 0;
            }
            if ( TestAllele1 == Allele12 ) {
            X12 = 1;
            }
            else {
            X12 = 0;
            }
            if ( TestAllele1 == Allele21 ) {
            X21 = 1;
            }
            else {
            X21 = 0;
            }
            if ( TestAllele1 == Allele22 ) {
            X22 = 1;
            }
            else {
            X22 = 0;
            }
            AvgXu = (X11 + X12 + X21 + X22)/4;
            //printf ( "allele %d: AvgXu = %f\n", e, AvgXu );
            AvgXiu =  (X11+X12)/2;
            //printf ( "allele %d: AvgXiu = %f\n", e, AvgXiu );
            AvgXiub = (X21+X22)/2;
            //printf ( "allele %d: AvgXiub = %f\n", e, AvgXiub );
            Term1 = pow((X11-AvgXiu),2) + pow((X12-AvgXiu),2) +
pow((X21-AvgXiub),2) + pow((X22-AvgXiub),2);
            //printf ( "allele %d: term1 = %f\n", e, Term1 );
            SSw+=Term1;
            Term2 = 2*(pow((AvgXiu-AvgXu),2) + pow((AvgXiub-
AvgXu),2));
            //printf ( "allele %d: term2 = %f\n", e, Term2 );
            SSb+=Term2;
        }
        Numerator = ( 2*(SSb)-SSw)*((N*(N-1))/2);
        Numerator2 = ( SSb*((N*(N-1))/2) );
        SumDenominator+=Denominator;
        SumDenominator2+=Denominator2;
        SumNumerator+=Numerator;
        SumNumerator2+=Numerator2;
```

```
        //printf( "Locus %d: Numerator = %f, Denominator = %f\n", d,
Numerator, Denominator );
        //printf( "Locus %d: Numerator2 = %f, Denominator2 = %f\n",
d, Numerator2, Denominator2 );

        }
        Ar = SumNumerator/SumDenominator;
        Ahat = (SumNumerator2/SumDenominator2) - 0.5;

            Row = DataRow[q];
        Column = DataCol[q];
        Row1 = DataRow[r];
        Column1 = DataCol[r];
        ArArray[pairwiseCounter] = Ar;
        AhatArray[pairwiseCounter] = Ahat;

        if ( Row >= Row1 ) {
        YSquare = pow((Row-Row1),2);
        }
        else
        YSquare = pow((Row1-Row),2);
        if ( Column >= Column1 ) {
        XSquare = pow((Column-Column1),2);
        }
        else
        XSquare = pow((Column1-Column),2);
        Distance = sqrt(XSquare+YSquare);
        DistanceArray[pairwiseCounter] = Distance;
        pairwiseCounter+=1;

    }
    }
}


void CountAlleles( int N, int L )
{
  int i,q,k,l,p;
  int Allele1, Allele2;
  int match1, match2;
  int TestAllele1;

  Alleles = calloc ( N*2, ( sizeof( int *)));
  if ( Alleles == NULL ) {
    printf( "Alleles 1st dimension memory allocation failure\n" );
  }
  for ( l = 0; l < N*2; l++ ) {
    Alleles[l] = calloc ( L, sizeof( int ) );
    if ( Alleles[l] == NULL ) {
      printf( "Alleles 2nd dimension memory allocation failure\n" );
    }
  }

  LocAlleleNumbers = ( int * ) calloc ( L, sizeof ( int ) );
  if ( LocAlleleNumbers == NULL )
    printf ( "LocAlleleNumbers memory allocation failure\n" );

  for ( i = 0; i < L; i++ ) {
    p = 0;
    for ( k = 0; k < N; k++ ) {
```

```
      Allele1 = Data1[k][i];
      Allele2 = Data2[k][i];
      if ( k == 0 ) {
    if ( Allele1 == Allele2 ) {
      Alleles[p][i] = Allele1;
      p+=1;
    }
    else {
      Alleles[p][i] = Allele1;
      p+=1;
      Alleles[p][i] = Allele2;
      p+=1;
    }
      }
      else {
    match1 = 0;
    match2 = 0;
    if ( Allele1 != Allele2 ) {
      for ( q = 0; q < p; q++ ) {
        TestAllele1 = Alleles[q][i];
        if ( TestAllele1 == Allele1 ) {
          match1 = 1;
        }
        if ( TestAllele1 == Allele2 ) {
          match2 = 1;
        }
      }
      if ( match1 == 0) {
        Alleles[p][i] = Allele1;
        p+=1;
      }
      if ( match2 == 0) {
        Alleles[p][i] = Allele2;
        p+=1;
      }
    }
    else if ( Allele1 == Allele2 ) {
      for ( q = 0; q < p; q++ ) {
        TestAllele1 = Alleles[q][i];
        if ( TestAllele1 == Allele1 ) {
          match1 = 1;
        }
      }
      if ( match1 == 0 ) {
        Alleles[p][i] = Allele1;
        p+=1;
      }
    }
      }
    }
    LocAlleleNumbers[i] = p;
    }
}


void SamplePopulationII ( int N, int NumRows, int NumCols, int L )
{

  int i,j,k,l,m,n;
  int RandomRow;
  int RandomCol;
```

```
int test, testAllele;
int testRow, testColumn;
int match;
int Allele1, Allele2;
int reps;

SampledRows = (int *) calloc ( N, ( sizeof( int ) ) );
if ( SampledRows == NULL ) {
  printf( "SampledRows memory failure\n" );
}

SampledColumns = (int *) calloc ( N, ( sizeof( int ) ) );
if ( SampledColumns == NULL ) {
  printf( "SampledColumns memory failure\n" );
}

DataRow = ( int * ) calloc ( N, ( sizeof ( int ) ) );
if ( DataRow == NULL ) {
  printf( "Data row memory allocation failure\n" );
}

DataCol = ( int * ) calloc (N, ( sizeof ( int ) ) );
if ( DataCol == NULL ) {
  printf( "Data col memory allocation failure\n" );
}

Data1 = calloc ( N, ( sizeof(int *)));
if ( Data1 == NULL ) {
  printf( "Data1 1st dimension memory allocation failure\n" );
}
for ( l = 0; l < N; l++ ) {
  Data1[l] = calloc ( L, sizeof( int ) );
  if ( Data1[l] == NULL ) {
    printf ( "Data1 2nd dimension memory allocation failure\n" );
  }
}

Data2 = calloc ( N, ( sizeof(int *)));
if ( Data2 == NULL ) {
  printf( "Data2 1st dimension memory allocation failure\n" );
}
for ( l = 0; l < N; l++ ) {
  Data2[l] = calloc ( L, sizeof( int ) );
  if ( Data2[l] == NULL ) {
    printf ( "Data2 2nd dimension memory allocation failure\n" );
  }
}

ActualN = 0;
reps = 0;
n = 0;
while ( ( ActualN < N ) && ( reps < 100*N) ) {
  test = 0;
  match = 1;
  if ( ActualN == 0 ) {
    while ( test == 0 ) {
  RandomRow = rand() % NumRows;
  RandomCol = rand() % NumCols;
  testAllele = Data[RandomRow][RandomCol][0][0];
  if ( testAllele > 0 ) {
    test = 1;
```

```
      SampledRows[ ActualN ] = RandomRow;
      SampledColumns[ ActualN ] = RandomCol;
      ActualN+=1;
    }
      }
    }
    else {
      while ( match == 1 ) {
    RandomRow = rand() % NumRows;
    RandomCol = rand() % NumCols;
    testAllele = Data[ RandomRow ][ RandomCol ][ 0 ][ 0 ];
    if ( testAllele > 0 ) {
      test = 1;
    }
    if ( test == 1 ) {
      match = 0;
      for ( j = 0; j < ActualN-1; j++ ) {
        testRow = SampledRows[ j ];
        testColumn = SampledColumns[ j ];
        if ( ( testRow == RandomRow ) && ( testColumn == RandomCol )
) {
          match = 1;
        }
      }
      if ( match == 0 ) {
        SampledRows[ ActualN ] = RandomRow;
        SampledColumns [ ActualN ] = RandomCol;
        ActualN+=1;
      }
    }
      }
    }

    for ( l = 0; l < L; l++ ) {
      Allele1 = Data [ RandomRow ][ RandomCol ][ l ][ 0 ];
      Allele2 = Data [ RandomRow ][ RandomCol ][ l ][ 1 ];
      //printf ( "Allele1 is %d\n", Allele1 );
      //printf ( "Allele2 is %d\n", Allele2 );
      Data1[ ActualN-1 ][l] = Allele1;
      Data2[ ActualN-1 ][l] = Allele2;
    }
    DataRow [ ActualN-1 ] = RandomRow;
    DataCol [ ActualN-1 ] = RandomCol;
    reps+=1;
  }
}

void SamplePopulation ( int N, int NumRows, int NumCols, int L, int
extent, int sigma )
{
    int i,j,k,l,m,n;
    int RandomRow;
    int RandomCol;
    int test, testAllele;
    int testRow, testColumn;
    int match;
    int Allele1, Allele2;
    int CentroidRow, CentroidCol;
    int DistanceR, DistanceC;
    int reps;
    int e;
```

```
    float DirectionR, DirectionC;


    e = floor( 0.5*extent*sigma );

    SampledRows = (int *) calloc ( N, ( sizeof( int ) ) );
    if ( SampledRows == NULL ) {
      printf( "SampledRows memory failure\n" );
    }

    SampledColumns = (int *) calloc ( N, ( sizeof( int ) ) );
    if ( SampledColumns == NULL ) {
    printf( "SampledColumns memory failure\n" );
    }

    DataRow = ( int * ) calloc ( N, ( sizeof ( int ) ) );
    if ( DataRow == NULL ) {
      printf( "Data row memory allocation failure\n" );
    }

    DataCol = ( int * ) calloc (N, ( sizeof ( int ) ) );
    if ( DataCol == NULL ) {
      printf( "Data col memory allocation failure\n" );
    }

    Data1 = calloc ( N, ( sizeof(int *)));
    if ( Data1 == NULL ) {
      printf( "Data1 1st dimension memory allocation failure\n" );
    }
    for ( l = 0; l < N; l++ ) {
      Data1[l] = calloc ( L, sizeof( int ) );
      if ( Data1[l] == NULL ) {
    printf ( "Data1 2nd dimension memory allocation failure\n" );
      }
    }

    Data2 = calloc ( N, ( sizeof(int *)));
    if ( Data2 == NULL ) {
      printf( "Data2 1st dimension memory allocation failure\n" );
    }
    for ( l = 0; l < N; l++ ) {
      Data2[l] = calloc ( L, sizeof( int ) );
      if ( Data2[l] == NULL ) {
    printf ( "Data2 2nd dimension memory allocation failure\n" );
      }
    }

    ActualN = 0;
    reps = 0;
    n = 0;
    while ( ( ActualN < N ) && ( reps < 100*N) ) { //possible that
may not find N samples within the sampling range specified by extent.
      test = 0;
      match = 1;
      if ( ActualN == 0 ) {
        while ( test == 0 ) {
          RandomRow = rand() % NumRows;
          RandomCol = rand() % NumCols;
          testAllele = Data[RandomRow][RandomCol][0][0];
          if ( testAllele > 0 ) {
```

```
          test = 1;
          CentroidRow = RandomRow;
          CentroidCol = RandomCol;
          SampledRows[ ActualN ] = RandomRow;
          SampledColumns[ ActualN ] = RandomCol;
          ActualN+=1;
            }
          }
        }
        else {
          while ( match == 1 ) {
            DistanceR = floor( (rand() % ( e )) + 1 );
            DistanceC = floor( (rand() % ( e )) + 1 );
            DirectionR = ( rand()/((double)RAND_MAX+1) );
            DirectionC = ( rand()/((double)RAND_MAX+1) );

            if ( DirectionR > 0.5 ) {
          RandomRow = CentroidRow+DistanceR;
          if ( RandomRow > NumRows-1 )
            RandomRow = CentroidRow-DistanceR;
            }
            else {
          RandomRow = CentroidRow-DistanceR;
          if ( RandomRow < 0 )
            RandomRow = CentroidRow+DistanceR;
            }

            if ( DirectionC > 0.5 ) {
          RandomCol = CentroidCol+DistanceC;
          if ( RandomCol > NumCols-1 )
            RandomCol = CentroidCol-DistanceC;
            }
            else {
          RandomCol = CentroidCol-DistanceC;
          if ( RandomCol < 0 )
            RandomCol = CentroidCol+DistanceC;
            }
            testAllele = Data[ RandomRow ][ RandomCol ][ 0 ][ 0 ];
            if ( testAllele > 0 ) {
          test = 1;
            }
            if ( test == 1 ) {
          match = 0;
          for ( j = 0; j < ActualN-1; j++ ) {
            testRow = SampledRows[ j ];
            testColumn = SampledColumns[ j ];
            if ( ( testRow == RandomRow ) && ( testColumn == RandomCol
) ) {
              match = 1;
            }
          }
          if ( match == 0 ) {
            SampledRows[ ActualN ] = RandomRow;
            SampledColumns [ ActualN ] = RandomCol;
            ActualN+=1;
          }
            }
          }
        }
        for ( l = 0; l < L; l++ ) {
          Allele1 = Data [ RandomRow ][ RandomCol ][ l ][ 0 ];
```

```
        Allele2 = Data [ RandomRow ][ RandomCol ][ l ][ 1 ];
        Data1[ ActualN-1 ][l] = Allele1;
        Data2[ ActualN-1 ][l] = Allele2;
      }
      DataRow [ ActualN-1 ] = RandomRow;
      DataCol [ ActualN-1 ] = RandomCol;
      reps+=1;
    }
}


void WriteDataMatrix ( int LatRows, int LatCols, int Loci )
{

    int i,j,k,l;
    int dataVal, dataVal2, sexVal;

    for ( i = 0; i < LatRows; i++ ) {
      for ( j = 0; j < LatCols; j++ ) {
        sexVal = sexTemp[i][j];
    sexData[i][j] = sexVal;
    for ( k = 0; k < Loci; k ++ ) {
      dataVal = dataTemp[i][j][k][0];
      dataVal2 = dataTemp[i][j][k][1];
      Data[i][j][k][0] = dataVal;
      Data[i][j][k][1] = dataVal2;
        }
      }
    }
    return;
}



void Reproduction ( int PopSize, int LatRows, int LatCols, int d, int
L, int year )
{

  int j, m;
  int occupied;
  int RandomRow1, RandomRow2;
  int RandomCol1, RandomCol2;
  int Test;
  int male, reps;
  int RandomMaleX, RandomMaleY;
  int empty;
  int counter;
  int RandomOffspringX, RandomOffspringY;
  int OffspringX, OffspringY;
  int FemaleAllele1, FemaleAllele2;
  int MaleAllele1, MaleAllele2;
  int RandomAllele1, RandomAllele2;
  int Sex;
  int XDistance, YDistance;

  float DirectionX, DirectionY;
  float MuRate;
  float RandomMu;
  float RandomMutate;
  float RandomSex;
  float Random;
```

```
  float DispersalDistance;
  float AvgDispersal;
  float *DispersalDistanceArray;

  DispersalDistanceArray = ( float * ) calloc ( PopSize, sizeof (
float ) );
  if ( DispersalDistanceArray == NULL )
    printf ( "DispersalDistanceArray memory allocation failure\n" );

  j = 0;
  while ( ( j < PopSize ) && ( j < LatRows*LatCols ) ) {
    occupied = 0;
    while ( occupied == 0 ) { /*Find a random female*/
      RandomRow1 = floor( rand() % LatRows );
      RandomCol1 = floor( rand() % LatCols );
      Test = Data [ RandomRow1 ][ RandomCol1 ][ 0 ][ 0 ];
      if ( Test != 0 ) {
    occupied = 1;
    Sex = sexData [ RandomRow1 ] [ RandomCol1 ];
    if ( Sex == 2 )
      occupied = 1;
    else
      occupied = 0;
      }
    }

    /*Find a random male within d units of the female*/
    male = 0;
    reps = 0;
    while ( ( male == 0) && ( reps < 1000) ) {
      DirectionX = ( rand()/((double)RAND_MAX+1) );
      DirectionY = ( rand()/((double)RAND_MAX+1) );
      RandomMaleX = floor( (rand() % d ) + 1);
      RandomMaleY = floor( (rand() % d ) + 1);
      if ( DirectionX > 0.5 ) {
    RandomRow2 = RandomRow1+RandomMaleX;
    if ( RandomRow2 > LatRows-1 ) {
      RandomRow2 = RandomRow1-RandomMaleX;
    }
      }
      else {
    RandomRow2 = RandomRow1-RandomMaleX;
    if ( RandomRow2 < 0 ) {
      RandomRow2 = RandomRow1+RandomMaleX;
    }
      }

      if ( DirectionY > 0.5 ) {
    RandomCol2 = RandomCol1+RandomMaleY;
    if ( RandomCol2 > LatCols-1 ) {
      RandomCol2 = RandomCol1-RandomMaleY;
    }
      }
      else {
    RandomCol2 = RandomCol1-RandomMaleY;
    if ( RandomCol2 < 0 ) {
      RandomCol2 = RandomCol1+RandomMaleY;
    }
      }
      Sex = sexData [ RandomRow2 ][ RandomCol2 ];
      if ( Sex == 1 )
```

```
    male = 1;
      reps+=1;
    }

  /*Find a new location for the single progeny*/
    if ( male == 1 ) {
      empty = 0;
      counter = 0;
      while ( ( empty == 0 ) && ( counter < 10000 ) ) {
    DirectionX = ( rand()/((double)RAND_MAX+1) );
    DirectionY = ( rand()/((double)RAND_MAX+1) );
    RandomOffspringX = floor(( rand() % d ) + 1);
    RandomOffspringY = floor(( rand() % d ) + 1);

    if ( DirectionX > 0.5 ) {
      OffspringX = RandomRow1+RandomOffspringX;
      if ( OffspringX > LatRows-1 ) {
        OffspringX = RandomRow1-RandomOffspringX;
      }
    }
    else {
      OffspringX = RandomRow1-RandomOffspringX;
      if ( OffspringX < 0 ) {
        OffspringX = RandomRow1+RandomOffspringX;
      }
    }

    if ( DirectionY > 0.5 ) {
      OffspringY = RandomCol1+RandomOffspringY;
      if ( OffspringY > LatCols-1 ) {
        OffspringY = RandomCol1-RandomOffspringY;
      }
    }
    else {
      OffspringY = RandomCol1-RandomOffspringY;
      if ( OffspringY < 0 ) {
        OffspringY = RandomCol1+RandomOffspringY;
      }
    }
    Test = dataTemp[OffspringX][OffspringY][0][0];
    if ( Test == 0 ) {
      empty = 1;
    }
    counter+=1;
      }

      if ( RandomRow1 >= OffspringX )
    XDistance = RandomRow1 - OffspringX;
      else
    XDistance = OffspringX-RandomRow1;

      if ( RandomCol1 >= OffspringY )
    YDistance = RandomCol1 - OffspringY;
      else
    YDistance = OffspringY-RandomCol1;
      DispersalDistance = sqrt( pow( XDistance, 2 ) + pow( YDistance,
2 ) );
      if ( WriteDispersalFile == 1 )
        fprintf ( ptrDispersalFile, "%f\n", DispersalDistance );
      DispersalDistanceArray [ j ] = DispersalDistance;
```

```
      RandomSex = rand()/((double)RAND_MAX+1);
      if ( RandomSex > 0.5 )
    Sex = 1;
      else
    Sex = 2;

      sexTemp [ OffspringX ][ OffspringY ] = Sex;

  /*Generate Alleles, through descent and mutation, and write to
dataTemp*/
      for ( m = 0; m < L; m++ ) {
    MuRate = MutationRate[m];
    FemaleAllele1 = Data[RandomRow1][RandomCol1][m][0];
    FemaleAllele2 = Data[RandomRow1][RandomCol1][m][1];
    MaleAllele1 = Data[RandomRow2][RandomCol2][m][0];
    MaleAllele2 = Data[RandomRow2][RandomCol2][m][1];

    RandomAllele1 = rand()/((double)RAND_MAX+1);
    RandomAllele2 = rand()/((double)RAND_MAX+1);
    if ( RandomAllele1 > 0.5 ) { /*choose allele1 from female*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          FemaleAllele1 = FemaleAllele1+1;
      }
        else if ( RandomMutate <= 0.5 ) {
          FemaleAllele1 = FemaleAllele1-1;
          if ( FemaleAllele1 <= 0 ) {
      FemaleAllele1 = FemaleAllele1+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][0] = FemaleAllele1;
    }
    else { /*choose allele2 from female*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          FemaleAllele2 = FemaleAllele2+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          FemaleAllele2 = FemaleAllele2-1;
          if ( FemaleAllele2 <= 0 ) {
      FemaleAllele2 = FemaleAllele2+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][0] = FemaleAllele2;
    }
    if ( RandomAllele2 > 0.5 ) { /*choose allele1 from male*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          MaleAllele1 = MaleAllele1+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          MaleAllele1 = MaleAllele1-1;
          if ( MaleAllele1 <= 0 ) {
```

```
        MaleAllele1 = MaleAllele1+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][1] = MaleAllele1;
    }
    else { /*choose allele2 from male*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          MaleAllele2 = MaleAllele2+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          MaleAllele2 = MaleAllele2-1;
          if ( MaleAllele2 <= 0 ) {
        MaleAllele2 = MaleAllele2+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][1] = MaleAllele2;
    }
      }
      j+=1;
    }
    if ( j == ( (LatRows*LatCols)-1 ) ) {
      Random = rand()/((double)RAND_MAX+1);
      NodesOccupied = floor( LatRows*LatCols*Random );
    }
  }

  AvgDispersal = mean ( DispersalDistanceArray, PopSize );
  MeanDispersal [ year ] = AvgDispersal;
  free ( DispersalDistanceArray );

}

void ReproductionII ( int PopSize, int LatRows, int LatCols, int dp,
float v, int L, int year )
{

  int j, m;
  int d;
  int occupied;
  int RandomRow1, RandomRow2;
  int RandomCol1, RandomCol2;
  int Test;
  int male, reps;
  int RandomMaleX, RandomMaleY;
  int empty;
  int counter;
  int RandomOffspringX, RandomOffspringY;
  int OffspringX, OffspringY;
  int FemaleAllele1, FemaleAllele2;
  int MaleAllele1, MaleAllele2;
  int RandomAllele1, RandomAllele2;
  int Sex;
  int XDistance, YDistance;

  float DirectionX, DirectionY;
  float MuRate;
```

```
  float RandomMu;
  float RandomMutate;
  float RandomSex;
  float Random;
  float DispersalDistance;
  float AvgDispersal;
  float *DispersalDistanceArray;
  float FloatVal;

  DispersalDistanceArray = ( float * ) calloc ( PopSize, sizeof (
float ) );
  if ( DispersalDistanceArray == NULL )
    printf ( "DispersalDistanceArray memory allocation failure\n" );

  j = 0;
  while ( ( j < PopSize ) && ( j < LatRows*LatCols ) ) {
    occupied = 0;
    while ( occupied == 0 ) { /*Find a random female*/
      RandomRow1 = floor( rand() % LatRows );
      RandomCol1 = floor( rand() % LatCols );
      Test = Data [ RandomRow1 ][ RandomCol1 ][ 0 ][ 0 ];
      if ( Test != 0 ) {
    occupied = 1;
    Sex = sexData [ RandomRow1 ] [ RandomCol1 ];
    if ( Sex == 2 )
      occupied = 1;
    else
      occupied = 0;
      }
    }

    //Draw a maximum dispersal distance from a normal distribution
with mean dp, var v
    FloatVal = 0;
    while ( FloatVal <= 0 ) {
      FloatVal = ( ( normal ( sqrt ( v ) ) ) + dp );
    }
    d = floor( FloatVal + 1 ); //for Random normal values of between
0 and 1 we add 1

    /*Find a random male within d units of the female*/
    male = 0;
    reps = 0;
    while ( ( male == 0) && ( reps < 1000) ) {
      DirectionX = ( rand()/((double)RAND_MAX+1) );
      DirectionY = ( rand()/((double)RAND_MAX+1) );
      RandomMaleX = floor( (rand() % d ) + 1);
      RandomMaleY = floor( (rand() % d ) + 1);
      if ( DirectionX > 0.5 ) {
    RandomRow2 = RandomRow1+RandomMaleX;
    if ( RandomRow2 > LatRows-1 ) {
      RandomRow2 = RandomRow1-RandomMaleX;
    }
      }
      else {
    RandomRow2 = RandomRow1-RandomMaleX;
    if ( RandomRow2 < 0 ) {
      RandomRow2 = RandomRow1+RandomMaleX;
    }
      }
```

```
      if ( DirectionY > 0.5 ) {
    RandomCol2 = RandomCol1+RandomMaleY;
    if ( RandomCol2 > LatCols-1 ) {
      RandomCol2 = RandomCol1-RandomMaleY;
    }
      }
      else {
    RandomCol2 = RandomCol1-RandomMaleY;
    if ( RandomCol2 < 0 ) {
      RandomCol2 = RandomCol1+RandomMaleY;
    }
      }
      Sex = sexData [ RandomRow2 ][ RandomCol2 ];
      if ( Sex == 1 )
    male = 1;
      reps+=1;
    }

  /*Find a new location for the single progeny*/

    if ( male == 1 ) {

      //Draw a maximum dispersal distance from a normal distribution
with mean dp, var v
      FloatVal = 0;
      while ( FloatVal <= 0 ) {
        FloatVal = ( ( normal ( sqrt ( v ) ) ) + dp );
      }
      d = floor( FloatVal + 1 ); //for Random normal values of
between 0 and 1 we add 1

      empty = 0;
      counter = 0;
      while ( ( empty == 0 ) && ( counter < 10000 ) ) {
    DirectionX = ( rand()/((double)RAND_MAX+1) );
    DirectionY = ( rand()/((double)RAND_MAX+1) );
    RandomOffspringX = floor(( rand() % d ) + 1);
    RandomOffspringY = floor(( rand() % d ) + 1);

    if ( DirectionX > 0.5 ) {
      OffspringX = RandomRow1+RandomOffspringX;
      if ( OffspringX > LatRows-1 ) {
        OffspringX = RandomRow1-RandomOffspringX;
      }
    }
    else {
      OffspringX = RandomRow1-RandomOffspringX;
      if ( OffspringX < 0 ) {
        OffspringX = RandomRow1+RandomOffspringX;
      }
    }

    if ( DirectionY > 0.5 ) {
      OffspringY = RandomCol1+RandomOffspringY;
      if ( OffspringY > LatCols-1 ) {
        OffspringY = RandomCol1-RandomOffspringY;
      }
    }
    else {
      OffspringY = RandomCol1-RandomOffspringY;
      if ( OffspringY < 0 ) {
```

```
        OffspringY = RandomCol1+RandomOffspringY;
      }
    }
    Test = dataTemp[OffspringX][OffspringY][0][0];
    if ( Test == 0 ) {
      empty = 1;
    }
    counter+=1;
      }

      if ( RandomRow1 >= OffspringX )
    XDistance = RandomRow1 - OffspringX;
      else
    XDistance = OffspringX-RandomRow1;

      if ( RandomCol1 >= OffspringY )
    YDistance = RandomCol1 - OffspringY;
      else
    YDistance = OffspringY-RandomCol1;
      DispersalDistance = sqrt( pow( XDistance, 2 ) + pow( YDistance,
2 ) );
      if ( WriteDispersalFile == 1 )
    fprintf ( ptrDispersalFile, "%f\n", DispersalDistance );
      DispersalDistanceArray [ j ] = DispersalDistance;

      RandomSex = rand()/((double)RAND_MAX+1);
      if ( RandomSex > 0.5 )
    Sex = 1;
      else
    Sex = 2;

      sexTemp [ OffspringX ][ OffspringY ] = Sex;

  /*Generate Alleles, through descent and mutation, and write to
dataTemp*/
      for ( m = 0; m < L; m++ ) {
    MuRate = MutationRate[m];
    FemaleAllele1 = Data[RandomRow1][RandomCol1][m][0];
    FemaleAllele2 = Data[RandomRow1][RandomCol1][m][1];
    MaleAllele1 = Data[RandomRow2][RandomCol2][m][0];
    MaleAllele2 = Data[RandomRow2][RandomCol2][m][1];

    RandomAllele1 = rand()/((double)RAND_MAX+1);
    RandomAllele2 = rand()/((double)RAND_MAX+1);
    if ( RandomAllele1 > 0.5 ) { /*choose allele1 from female*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          FemaleAllele1 = FemaleAllele1+1;
      }
        else if ( RandomMutate <= 0.5 ) {
          FemaleAllele1 = FemaleAllele1-1;
          if ( FemaleAllele1 <= 0 ) {
        FemaleAllele1 = FemaleAllele1+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][0] = FemaleAllele1;
    }
    else { /*choose allele2 from female*/
```

```
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          FemaleAllele2 = FemaleAllele2+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          FemaleAllele2 = FemaleAllele2-1;
          if ( FemaleAllele2 <= 0 ) {
        FemaleAllele2 = FemaleAllele2+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][0] = FemaleAllele2;
    }
    if ( RandomAllele2 > 0.5 ) { /*choose allele1 from male*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          MaleAllele1 = MaleAllele1+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          MaleAllele1 = MaleAllele1-1;
          if ( MaleAllele1 <= 0 ) {
        MaleAllele1 = MaleAllele1+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][1] = MaleAllele1;
    }
    else { /*choose allele2 from male*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          MaleAllele2 = MaleAllele2+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          MaleAllele2 = MaleAllele2-1;
          if ( MaleAllele2 <= 0 ) {
        MaleAllele2 = MaleAllele2+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][1] = MaleAllele2;
    }
      }
      j+=1;
    }
    if ( j == ( (LatRows*LatCols)-1 ) ) {
      Random = rand()/((double)RAND_MAX+1);
      NodesOccupied = floor( LatRows*LatCols*Random );
    }
  }

  AvgDispersal = mean ( DispersalDistanceArray, PopSize );
  MeanDispersal [ year ] = AvgDispersal;
  free ( DispersalDistanceArray );

}
```

```
void ReproductionIII ( int PopSize, int LatRows, int LatCols, int d1,
float v1, int d2, float v2, int L, int year )
{

  int d;
  int j, m;
  int occupied;
  int RandomRow1, RandomRow2;
  int RandomCol1, RandomCol2;
  int Test;
  int male, reps;
  int RandomMaleX, RandomMaleY;
  int empty;
  int counter;
  int RandomOffspringX, RandomOffspringY;
  int OffspringX, OffspringY;
  int FemaleAllele1, FemaleAllele2;
  int MaleAllele1, MaleAllele2;
  int RandomAllele1, RandomAllele2;
  int Sex;
  int XDistance, YDistance;

  float DirectionX, DirectionY;
  float MuRate;
  float RandomMu;
  float RandomMutate;
  float RandomSex;
  float Random;
  float DispersalDistance;
  float AvgDispersal;
  float *DispersalDistanceArray;
  float FloatVal;

  DispersalDistanceArray = ( float * ) calloc ( PopSize, sizeof (
float ) );
  if ( DispersalDistanceArray == NULL )
    printf ( "DispersalDistanceArray memory allocation failure\n" );

  j = 0;
  while ( ( j < PopSize ) && ( j < LatRows*LatCols ) ) {
    occupied = 0;
    while ( occupied == 0 ) { /*Find a random female*/
      RandomRow1 = floor( rand() % LatRows );
      RandomCol1 = floor( rand() % LatCols );
      Test = Data [ RandomRow1 ][ RandomCol1 ][ 0 ][ 0 ];
      if ( Test != 0 ) {
    occupied = 1;
    Sex = sexData [ RandomRow1 ] [ RandomCol1 ];
    if ( Sex == 2 )
      occupied = 1;
    else
      occupied = 0;
      }
    }

    FloatVal = 0;
    while ( FloatVal <= 0 ) {
      Random = rand()/((double)RAND_MAX+1);
      if ( Random >= 0.05 ) {
    FloatVal = ( ( normal ( sqrt ( v1 ) ) ) + d1 );
```

```
    }
    else if ( Random < 0.05 ) {
  FloatVal = ( ( normal ( sqrt ( v2 ) ) ) + d2 );
    }
  }
  d = floor( FloatVal + 1);


  /*Find a random male within d units of the female*/
  male = 0;
  reps = 0;
  while ( ( male == 0) && ( reps < 1000) ) {
    DirectionX = ( rand()/((double)RAND_MAX+1) );
    DirectionY = ( rand()/((double)RAND_MAX+1) );
    RandomMaleX = floor( (rand() % d ) + 1);
    RandomMaleY = floor( (rand() % d ) + 1);
    if ( DirectionX > 0.5 ) {
  RandomRow2 = RandomRow1+RandomMaleX;
  if ( RandomRow2 > LatRows-1 ) {
    RandomRow2 = RandomRow1-RandomMaleX;
  }
    }
    else {
  RandomRow2 = RandomRow1-RandomMaleX;
  if ( RandomRow2 < 0 ) {
    RandomRow2 = RandomRow1+RandomMaleX;
  }
    }

    if ( DirectionY > 0.5 ) {
  RandomCol2 = RandomCol1+RandomMaleY;
  if ( RandomCol2 > LatCols-1 ) {
    RandomCol2 = RandomCol1-RandomMaleY;
  }
    }
    else {
  RandomCol2 = RandomCol1-RandomMaleY;
  if ( RandomCol2 < 0 ) {
    RandomCol2 = RandomCol1+RandomMaleY;
  }
    }
    Sex = sexData [ RandomRow2 ][ RandomCol2 ];
    if ( Sex == 1 )
  male = 1;
    reps+=1;
  }

/*Find a new location for the single progeny*/
  if ( male == 1 ) {
    FloatVal = 0;
    while ( FloatVal <= 0 ) {
  Random = rand()/((double)RAND_MAX+1);
  if ( Random >= 0.05 ) {
    FloatVal = ( ( normal ( sqrt ( v1 ) ) ) + d1 );
  }
  else if ( Random < 0.05 ) {
    FloatVal = ( ( normal ( sqrt ( v2 ) ) ) + d2 );
  }
    }
    d = floor( FloatVal + 1);
```

```
      empty = 0;
      counter = 0;
      while ( ( empty == 0 ) && ( counter < 10000 ) ) {
    DirectionX = ( rand()/((double)RAND_MAX+1) );
    DirectionY = ( rand()/((double)RAND_MAX+1) );
    RandomOffspringX = floor(( rand() % d ) + 1);
    RandomOffspringY = floor(( rand() % d ) + 1);

    if ( DirectionX > 0.5 ) {
      OffspringX = RandomRow1+RandomOffspringX;
      if ( OffspringX > LatRows-1 ) {
        OffspringX = RandomRow1-RandomOffspringX;
      }
    }
    else {
      OffspringX = RandomRow1-RandomOffspringX;
      if ( OffspringX < 0 ) {
        OffspringX = RandomRow1+RandomOffspringX;
      }
    }

    if ( DirectionY > 0.5 ) {
      OffspringY = RandomCol1+RandomOffspringY;
      if ( OffspringY > LatCols-1 ) {
        OffspringY = RandomCol1-RandomOffspringY;
      }
    }
    else {
      OffspringY = RandomCol1-RandomOffspringY;
      if ( OffspringY < 0 ) {
        OffspringY = RandomCol1+RandomOffspringY;
      }
    }
    Test = dataTemp[OffspringX][OffspringY][0][0];
    if ( Test == 0 ) {
      empty = 1;
    }
    counter+=1;
      }

      if ( RandomRow1 >= OffspringX )
    XDistance = RandomRow1 - OffspringX;
      else
    XDistance = OffspringX-RandomRow1;

      if ( RandomCol1 >= OffspringY )
    YDistance = RandomCol1 - OffspringY;
      else
    YDistance = OffspringY-RandomCol1;
      DispersalDistance = sqrt( pow( XDistance, 2 ) + pow( YDistance,
2 ) );
      if ( WriteDispersalFile == 1 )
    fprintf ( ptrDispersalFile, "%f\n", DispersalDistance );
      DispersalDistanceArray [ j ] = DispersalDistance;

      RandomSex = rand()/((double)RAND_MAX+1);
      if ( RandomSex > 0.5 )
    Sex = 1;
      else
    Sex = 2;
```

```
      sexTemp [ OffspringX ][ OffspringY ] = Sex;

  /*Generate Alleles, through descent and mutation, and write to
dataTemp*/
      for ( m = 0; m < L; m++ ) {
    MuRate = MutationRate[m];
    FemaleAllele1 = Data[RandomRow1][RandomCol1][m][0];
    FemaleAllele2 = Data[RandomRow1][RandomCol1][m][1];
    MaleAllele1 = Data[RandomRow2][RandomCol2][m][0];
    MaleAllele2 = Data[RandomRow2][RandomCol2][m][1];

    RandomAllele1 = rand()/((double)RAND_MAX+1);
    RandomAllele2 = rand()/((double)RAND_MAX+1);
    if ( RandomAllele1 > 0.5 ) { /*choose allele1 from female*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          FemaleAllele1 = FemaleAllele1+1;
      }
        else if ( RandomMutate <= 0.5 ) {
          FemaleAllele1 = FemaleAllele1-1;
          if ( FemaleAllele1 <= 0 ) {
        FemaleAllele1 = FemaleAllele1+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][0] = FemaleAllele1;
    }
    else { /*choose allele2 from female*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          FemaleAllele2 = FemaleAllele2+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          FemaleAllele2 = FemaleAllele2-1;
          if ( FemaleAllele2 <= 0 ) {
        FemaleAllele2 = FemaleAllele2+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][0] = FemaleAllele2;
    }
    if ( RandomAllele2 > 0.5 ) { /*choose allele1 from male*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          MaleAllele1 = MaleAllele1+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          MaleAllele1 = MaleAllele1-1;
          if ( MaleAllele1 <= 0 ) {
        MaleAllele1 = MaleAllele1+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][1] = MaleAllele1;
    }
```

```
    else { /*choose allele2 from male*/
      RandomMu = rand()/((double)RAND_MAX+1);
      if ( MuRate > RandomMu ) {
        RandomMutate = rand()/((double)RAND_MAX+1);
        if ( RandomMutate > 0.5 ) {
          MaleAllele2 = MaleAllele2+1;
        }
        else if ( RandomMutate <= 0.5 ) {
          MaleAllele2 = MaleAllele2-1;
          if ( MaleAllele2 <= 0 ) {
        MaleAllele2 = MaleAllele2+2;
          }
        }
      }
      dataTemp[OffspringX][OffspringY][m][1] = MaleAllele2;
    }
      }
      j+=1;
    }
    if ( j == ( (LatRows*LatCols)-1 ) ) {
      Random = rand()/((double)RAND_MAX+1);
      NodesOccupied = floor( LatRows*LatCols*Random );
    }
  }

  AvgDispersal = mean ( DispersalDistanceArray, PopSize );
  MeanDispersal [ year ] = AvgDispersal;
  free ( DispersalDistanceArray );

}


void AssignMuRate ( float a, float b, int L )
{
    int i;
    float testAlpha, mu;
    float XGamma;

    testAlpha = a - floor( a );
    for ( i = 0; i < L; i++ ) {
    if ( testAlpha == 0 ) {
        XGamma = gammaMultiplication( a );
    }
    else if ( ( testAlpha != 0 ) && ( a > 0 ) && ( a < 1 ) ) {
        XGamma = gammaRejection( a );
    }
    else if ( ( testAlpha != 0) && ( a > 1 ) ) {
        XGamma = gammaTwoPart( a );
    }
    mu = ( XGamma/a )*( b );
    *(MutationRate+i) = mu;
    }
}

void InitialiseArrays ( int LatRows, int LatCols, int N, int L, int
Y, int Pops )
{
  int i, j, k, l;

  MutationRate = ( float *) calloc ( L, sizeof ( float ) );
  if ( MutationRate == NULL )
```

```
  printf( "Mutation rate memory allocation failure\n" );

MeanDispersal = ( float *) calloc ( Y, sizeof ( float ) );
if ( MeanDispersal == NULL ) {
  printf ( "MeanDispersal memory allocation failure\n" );
}

NbSizeAr = ( float * ) calloc ( Y, sizeof ( float ) );
if ( NbSizeAr == NULL ) {
  printf ( "NbSizeAr memory allocation failure\n" );
}

NbSizeArSD = ( float * ) calloc ( Y, sizeof ( float ) );
if ( NbSizeArSD == NULL ) {
  printf ( "NbSizeArSD memory allocation failure\n" );
}

NbSizeAhat = ( float * ) calloc ( Y, sizeof ( float ) );
if ( NbSizeAhat == NULL ) {
  printf ( "NbSizeAhat memory allocation failure\n" );
}

NbSizeAhatSD = ( float * ) calloc ( Y, sizeof ( float ) );
if ( NbSizeAhatSD == NULL ) {
  printf ( "NbSizeAhatSD memory allocation failure\n" );
}

PopSizeArray = ( int * ) calloc ( Y, sizeof ( int ) );
if ( PopSizeArray == NULL ) {
  printf ( "PopSizeArray memory allocation failure\n" );
}

TotalDiv = ( float * ) calloc (Y, ( sizeof ( float ) ) );
if ( TotalDiv == NULL ) {
  printf( "TotalDiv memory allocation failure\n" );
}

TotalDivSD = ( float * ) calloc (Y, sizeof ( float ) );
if ( TotalDivSD == NULL ) {
  printf( "TotalDivVar memory allocation failure\n" );
}

TotalAllelicDiv = ( float * ) calloc (Y, sizeof ( float ) );
if ( TotalAllelicDiv == NULL ) {
  printf( "TotalAllelicDiv memory allocation failure\n" );
}

TotalAllelicDivSD = ( float * ) calloc (Y, sizeof ( float ) );
if ( TotalAllelicDivSD == NULL ) {
  printf( "TotalAllelicDivSD memory allocation failure\n" );
}

AdiArray = ( float * ) calloc (Y, sizeof ( float ) );
if ( AdiArray == NULL ) {
  printf( "AdiArray memory allocation failure\n" );
}

AdiSDArray = ( float * ) calloc (Y, sizeof ( float ) );
if ( AdiSDArray == NULL ) {
  printf( "AdiSDArray memory allocation failure\n" );
}
```

```
  if ( WriteSharedAlleleDistances == 1 ) {
    SharedDistances = calloc ( ( ( L*N*(N-1) )/2 ), sizeof( float **
) );
    if ( SharedDistances == NULL ) {
      printf ( "1st dimension memory allocation failure:
SharedDistances\n" );
    }
    for ( i = 0; i < ( (L*N*(N-1))/2 ); i++ ) {
      SharedDistances[ i ] = calloc ( Y, sizeof( float *) );
      if ( SharedDistances[ i ] == NULL )
    printf ( "2nd dimension memory allocation failure:
SharedDistances\n" );
      for ( j = 0; j < Y; j++ ) {
    SharedDistances[ i ][ j ] = calloc ( Pops, sizeof( float ) );
    if ( SharedDistances[ i ][ j ] == NULL )
      printf( "3rd dimension memory allocation failure:
SharedDistances\n" );
      }
    }
  }

  Data = calloc ( LatRows, sizeof( int *** ) );
  if ( Data == NULL ) {
    printf ( "1st dimension memory allocation failure: Data\n" );
  }
  for ( i = 0; i < LatRows; i++ ) {
    Data[ i ] = calloc ( LatCols, sizeof( int ** ) );
    if ( Data[ i ] == NULL ) {
      printf ( "2nd dimension memory allocation failure: Data\n" );
    }
    for ( j = 0; j < LatCols; j++ ) {
      Data[ i ][ j ] = calloc ( L, sizeof( int * ) );
      if ( Data [ i ][ j ] == NULL ) {
    printf ( "3rd dimension memory allocation failure: Data\n" );
      }
      for ( k = 0; k < L; k++ ) {
    Data[ i ][ j ][ k ] = calloc ( 2, sizeof( int ) );
    if ( Data [ i ][ j ][ k ] == NULL ) {
      printf ( "4th dimension memory allocation failure: Data\n" );
    }
      }
    }
  }

  sexData = calloc ( LatRows, sizeof( int * ) );
  if ( sexData == NULL ) {
    printf( "1st dimension memory allocation failure: sexData\n" );
  }
  for ( i = 0; i < LatRows; i++ ) {
    sexData[ i ] = calloc( LatCols, sizeof( int ) );
    if ( sexData[ i ] == NULL ) {
      printf ( "2nd dimension memory allocation failure: sexData\n"
);
    }
  }



}
```

```
void ResetTempArrays ( int LatRows, int LatCols, int L )
{
    int i, j, k, l;

    for ( i = 0; i <= LatRows-1; i++ ) {
    for ( j = 0; j <= LatCols-1; j++ ) {
        sexTemp[i][j] = 0;
        for ( k = 0; k <= L-1; k++ ) {
        for ( l = 0; l < 2; l++ ) {
            dataTemp[i][j][k][l] = 0;
        }
        }
    }
    }
    return;
}

void CreateTempArrays ( int LatRows, int LatCols, int L )
{

  int i, j, k, l;

    dataTemp = calloc (LatRows, sizeof(int ***));
    if ( dataTemp == NULL ) {
      printf( "1st dimension memory allocation (dataTemp) failure\n"
);
    }
    for ( i = 0; i < LatRows; i++ ) {
      dataTemp[i] = calloc ( LatCols, sizeof(int **));
      if ( dataTemp[i] == NULL ) {
        printf( "2nd dimension memory allocation (dataTemp) failure
at %d\n", i );
      }
      for ( j = 0; j < LatCols; j++ ) {
        dataTemp[i][j] = calloc ( L, sizeof(int *));
    if ( dataTemp[i][j] == NULL ) {
          printf( "3rd dimension memory allocation (dataTemp) failure
at %d\t%d\n", i, j );
    }
    for ( k = 0; k < L; k++ ) {
          dataTemp[i][j][k] = calloc (2, sizeof(int));
          if ( dataTemp[i][j][k] == NULL ) {
            printf( "4th dimension memory allocation (dataTemp)
failure at %d\t%d\t%d\n", i, j, k );
          }
        }
      }
    }

    sexTemp = calloc (LatRows, sizeof(int *));
    if ( sexTemp == NULL ) {
      printf( "1st dimension memory allocation failure, sexTemp\n" );
    }
    for ( i = 0; i < LatRows; i++ ) {
      sexTemp[i] = calloc (LatCols, sizeof(int));
      if ( sexTemp[i] == NULL ) {
        printf( "2nd dimension memory allocation failure, sexTemp\n"
);
      }
    }
    return;
```

```
}


int SetStartingConditions ( int NumLoci, int LatticeRows, int
LatticeCols, int MeanNumAllelesPerLoc, float alleleAlpha, float het )
{
  int a,i,j,k,l,m,n,o;
  int NodeCol, NodeRow;
  int match;
  int testRow, testCol;
  int Alleles;
  int *LocAlleleNumbers;
  int Allele1, Allele2;
  int StartPopSize;

  float testAlpha;
  float XGamma;
  float RandomNum;

  LocAlleleNumbers = (int *) calloc ( NumLoci, ( sizeof(int)));
  if ( LocAlleleNumbers == NULL ) {
    printf( "LocAlleleNumbers memory failure\n" );
  }

  testAlpha = alleleAlpha - floor( alleleAlpha );
  for ( a = 0; a <= NumLoci-1; a++ ) {
    Alleles = 0;
    while ( ( Alleles < 2 ) ) {
      if ( testAlpha == 0 ) {
    XGamma = gammaMultiplication( alleleAlpha );
      }
      else if ( ( testAlpha != 0 ) && (alleleAlpha > 0 ) && (
alleleAlpha < 1 ) ) {
    XGamma = gammaRejection( alleleAlpha );
      }
      else if ( ( testAlpha != 0) && ( alleleAlpha > 1 ) ) {
    XGamma = gammaTwoPart( alleleAlpha );
      }
      Alleles = (XGamma/alleleAlpha)*(MeanNumAllelesPerLoc);
      *(LocAlleleNumbers+a) = Alleles;
    }
  }
  printf( "Number of Alleles per locus\n" );
  for ( l = 0; l < NumLoci; l++ ) {
    printf ("Loc%d\t", l+1 );
  }
  printf ( "\n");

  for ( l = 0; l <= NumLoci-1; l++ ) {
    Alleles = *(LocAlleleNumbers+l);
    printf( "%d\t", Alleles );
  }
  printf( "\n" );

  NodesOccupied = floor(LatticeRows*LatticeCols*percentOccupied);
  printf( "Nodes occupied is %d\n", NodesOccupied );

  NodeRows = (int *) calloc (NodesOccupied, ( sizeof(int)));
  if ( NodeRows == NULL ) {
    printf( "NodeRows memory failure\n" );
  }
```

```
  NodeCols = (int *) calloc (NodesOccupied, ( sizeof(int)));
  if ( NodeCols == NULL ) {
    printf( "NodeCols memory failure\n" );
  }

  for ( i = 0; i < NodesOccupied; i++ ) {
    match = 1;
    while ( match == 1) {
      NodeCol = ( rand() % (LatticeCols) );
      NodeRow = ( rand() % (LatticeRows) );
      if ( i == 0 ) {
    NodeRows[i] = NodeRow;
    NodeCols[i] = NodeCol;
    match = 0;
      }

      else if ( i > 0) {
    j = 0;
    match = 0;
    while ( ( j <= i-1) ) {
      testRow = NodeRows[j];
      testCol = NodeCols[j];
      if ( ( testRow == NodeRow) && ( testCol == NodeCol ) ) {
        match = 1;
      }
      j+=1;
    }

    if ( match == 0 ) {
      NodeRows[i] = NodeRow;
      NodeCols[i] = NodeCol;
    }
      }
    }

    for ( k = 0; k <= NumLoci-1; k++ ) {
      Alleles = *(LocAlleleNumbers+k);
      RandomNum = rand()/((double)RAND_MAX+1);
      if ( RandomNum <= het ) {
    Allele1 = (rand() % Alleles)+1;
    Allele2 = Allele1;
    while ( Allele1 == Allele2 ) {
      Allele2 = (rand() % Alleles)+1;
    }
    Data[NodeRow][NodeCol][k][0] = Allele1;
    Data[NodeRow][NodeCol][k][1] = Allele2;
      }
      else if ( RandomNum > het ) {
    Allele1 = (rand() % Alleles)+1;
    Data[NodeRow][NodeCol][k][0] = Allele1;
    Data[NodeRow][NodeCol][k][1] = Allele2;
      }
    }
  }
  return;
}

int AssignSex ( int NodesOccupied, int LatRows, int LatCols)
{
```

```
    int j;
    int NodeCol, NodeRow;
    int Sex;
    float RandomSex;
    int SizeMem;

    for ( j = 0; j < NodesOccupied; j++ ) {
    NodeRow = NodeRows[j];
    NodeCol = NodeCols[j];
        RandomSex = ( rand()/((double)RAND_MAX+1) );
      if ( RandomSex > 0.5 ) {
        Sex = 1; /*ie male*/
      }
      else {
        Sex = 2; /*ie female*/
      }
      sexData[NodeRow][NodeCol] = Sex;
    }
    return;
}

/* Gamma functions */
/*All gamma functions are from Ahrens & Dieter 1974. Computer methods
for sampling from Gamma, Beta, Poisson and Binomial distributions.
Computing 12: 223-246.*/

 float gammaMultiplication ( float x )
/*When alpha is an integer*/
     {
     int j;
     float p, RandomU, alphaInt;

     alphaInt = floor(x);
     j = 1;
     p = j;
     RandomU = rand()/((double)RAND_MAX+1);
     p = p*RandomU;
     while ( j != alphaInt ) {
         RandomU = rand()/((double)RAND_MAX+1);
         p = p*RandomU;
         j+=1;
     }
     return -( log(p) ) ;
     }

 float gammaRejection ( float x )
/*when alpha is real and <1*/
     {
     float RandomU, RandomU2, FuncB, FuncP, xGamma;

     RandomU = rand()/((double)RAND_MAX+1);
     FuncB = ( ( exp(1) + x)/exp(1) );
     FuncP = FuncB*RandomU;
     if (FuncP > 1 ) {
         xGamma = -log( ( FuncB-FuncP)/x );
         RandomU2 = rand()/((double)RAND_MAX+1);
     }
     else {
         xGamma = pow( FuncP, (1/x) );
         RandomU2 = rand()/((double)RAND_MAX+1);
     }
```

```
     while ( ( RandomU2 > exp (-xGamma) ) || ( RandomU2 > pow
(xGamma, (x-1) ) ) ) {
             RandomU = rand()/((double)RAND_MAX+1);
          FuncB = ( ( exp(1) + x)/exp(1) );
          FuncP = FuncB*RandomU;
     if (FuncP > 1 ) {
          xGamma = -log( ( FuncB-FuncP)/x );
          RandomU2 = rand()/((double)RAND_MAX+1);
     }
     else {
          xGamma = pow( FuncP, (1/x) );
          RandomU2 = rand()/((double)RAND_MAX+1);
     }
     }
     return xGamma;
     }


 float gammaTwoPart ( float x )
/*when alpha is real and >1*/
     {
     int j;
          float alphaInt, p, RandomU, RandomU2;
     float xGammaTwoPart1;
     float xGammaTwoPart2;
     float FuncB, FuncP;

     alphaInt = floor ( x );
     j = 1;
     p = j;
     RandomU = rand()/((double)RAND_MAX+1);
     p = p*RandomU;
     while ( j != alphaInt ) {
          RandomU = rand()/((double)RAND_MAX+1);
          p = p*RandomU;
          j+=1;
     }
     xGammaTwoPart1 = -log(p);

          RandomU = rand()/((double)RAND_MAX+1);
     FuncB = ( ( exp(1) + x)/exp(1) );
     FuncP = FuncB*RandomU;
     if (FuncP > 1 ) {
          xGammaTwoPart2 = -log( ( FuncB-FuncP)/x );
          RandomU2 = rand()/((double)RAND_MAX+1);
     }
     else {
          xGammaTwoPart2 = pow( FuncP, (1/x) );
          RandomU2 = rand()/((double)RAND_MAX+1);
     }

     while ( ( RandomU2 > ( exp (-xGammaTwoPart2) ) ) || ( RandomU2 >
pow (xGammaTwoPart2, (x-1) ) ) ) {
             RandomU = rand()/((double)RAND_MAX+1);
          FuncB = ( ( exp(1) + x)/exp(1) );
          FuncP = FuncB*RandomU;
          if (FuncP > 1 ) {
            xGammaTwoPart2 = -log( ( FuncB-FuncP)/x );
            RandomU2 = rand()/((double)RAND_MAX+1);
          }
          else {
```

```
            xGammaTwoPart2 = pow( FuncP, (1/x) );
            RandomU2 = rand()/((double)RAND_MAX+1);
          }
       }
       return xGammaTwoPart1 + xGammaTwoPart2;
     }

float normal ( float sd )
{

    float normalX;
    float u1, u2;

    u1 = rand()/((double)RAND_MAX+1);
    u2 = rand()/((double)RAND_MAX+1);
    normalX = sqrt(-2*log(u1))*sin(2*PI*u2);
    normalX = normalX*sd;
    return normalX;

}

float sumofsquares ( float a[], float Y, int x)
{
  float value=0, SumSquares=0;
  int i;

  for ( i = 0; i <= x-1; i ++) {
    value = pow( (a [ i ] - Y), 2 );
    SumSquares+=value;
  }
  return SumSquares;
}

float mean ( float a[], int x )
{
  float value=0;
  float Sum=0;
  float average=0;
  int i;

  for ( i = 0; i < x; i++ ) {
    value = a [ i ];
    Sum+=value;
  }
  average = Sum/(double)x;
  return average;
}
```

Appendix III

CoalFace Kylix/Delphi Source Code

"…tracing the ancestry of a gene backward in time and building up the family tree of the genes (at a particular locus) in a population sample back to the point at which they have a single common ancestor."

Kingman 2000

```
unit Unit1;

interface

uses
  SysUtils, Types, Classes, Variants, QMenus, QTypes, QGraphics, QControls,
QForms,
  QDialogs, QStdCtrls, QExtCtrls, Math, QComCtrls, QButtons, QFileCtrls;

type
  TfrmRandCoal = class(TForm)
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    lblMutationRateExpl: TLabel;
    lblMutationRate: TLabel;
    lblPopulationSize: TLabel;
    lblSampleSize: TLabel;
    edtSampleSize: TEdit;
    edtPopulationSize: TEdit;
    edtMutationRate: TEdit;
    chkScatterMut: TCheckBox;
    chkSegSites: TCheckBox;
    lblSequenceLength: TLabel;
    edtSeqLength: TEdit;
    chkSeqInput: TCheckBox;
    chkShapeParameter: TCheckBox;
    chkInvarSites: TCheckBox;
    edtAlpha: TEdit;
    edtSeqFile: TEdit;
    edtI: TEdit;
    chkMultipleSims: TCheckBox;
    edtNumberSimulations: TEdit;
    dlgSaveFile: TSaveDialog;
    btnRunSimulation: TButton;
    lblDNAParams: TLabel;
    lblDNASeqParam: TLabel;
    Label6: TLabel;
    Image1: TImage;
    chkDrawMut: TCheckBox;
    edtFreqA: TEdit;
    edtFreqC: TEdit;
    edtFreqG: TEdit;
    lblFreqA: TLabel;
    lblFreqC: TLabel;
    lblFreqG: TLabel;
    TabSheet4: TTabSheet;
    lblFactorsN: TLabel;
    chkVarRepSuc: TCheckBox;
```

chkFluctuatingN: TCheckBox;
edtVarRepSuc: TEdit;
edtGrowthStart: TEdit;
lblGrowthStart: TLabel;
lblGrowthEnd: TLabel;
edtGrowthEnd: TEdit;
lblFlucN: TLabel;
edtChangeN: TEdit;
radJC69: TRadioButton;
radF81: TRadioButton;
radK2P: TRadioButton;
radHKY85: TRadioButton;
lblBaseFrequencies: TLabel;
lblTiTvRatio: TLabel;
edtTiTv: TEdit;
rdgOutput: TRadioGroup;
lblMsatParam: TLabel;
lblNoLoci: TLabel;
edtNumLoci: TEdit;
radInfiniteAlleles: TRadioButton;
radStepwise: TRadioButton;
Label1: TLabel;
edtMsatIn: TEdit;
chkMsats: TCheckBox;
radAllele: TRadioButton;
TabSheet5: TTabSheet;
lblGeneral: TLabel;
lblNexusTrees: TLabel;
lblTMRCA: TLabel;
lblSeqDataOut: TLabel;
edtSeqOut: TEdit;
lblMsatOut: TLabel;
edtTmrca: TEdit;
edtMSatData: TEdit;
edtNewickTrees: TEdit;
chkArlequin: TCheckBox;
lblLogFile: TLabel;
edtLogFile: TEdit;
ProgressBar1: TProgressBar;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Image3: TImage;
Label7: TLabel;
chkDiploid: TCheckBox;
chkSeqOut: TCheckBox;
chkMSatOut: TCheckBox;
lblMutDistrib: TLabel;
lblCoalDistrib: TLabel;

```
    edtMutDistrib: TEdit;
    edtCoalDistrib: TEdit;
    memSimNum: TMemo;
    chkSeqDiversity: TCheckBox;
    chkMSatDiversity: TCheckBox;
    lblOutDir: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Label8: TLabel;
    Label9: TLabel;
    memDir: TMemo;
    BitBtn3: TBitBtn;
    DirectoryTreeView1: TDirectoryTreeView;
    edtMeanOffsp: TEdit;
    Label10: TLabel;

    procedure RunSimulation(Sender: TObject);
    //procedure Scale(Sender: TObject);

    procedure SaveGenealogy(Sender: TObject);
    procedure CreateForm(Sender: TObject);
    procedure AssignDirectory(Sender: TObject);
    procedure TreeFwd(Sender: TObject);
    procedure TreeBack(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmRandCoal : TfrmRandCoal;
  RandomkArray : array of integer;
  CountCoalesce : integer;
  CountCoalesce2 : integer;
  CountMutation : integer;
  NoMutations : integer;
  SortedRandArray : array of integer;
  CoalesceTime : integer;
  SampleSize : integer;
  TreeMatrix : array of array of integer;
  MutationMatrix : array of array of integer;
  BranchLengths : array of array of integer;
  CoalGenerations : array of array of integer;
  MutationScatter : array of array of integer;
  NewickArray : array of array of string;
  NewickArray2 : array of array of string;
  Data : array of array of char;
  DiploidData : array of array of char;
```

```
MData : array of array of integer;
DivInd : array of array of real;
NumberMutationArray : array of integer;
CoalArrayLength : integer;
N : integer;
NumberSimulations : integer;
outfile: textfile;
outfileN : Textfile;
outfileTMRCA : Textfile;
outfileCoalDis : Textfile;
outfileMuDis : Textfile;
GammaOut : textfile;
GammaOutPerSite : textfile;
ArlequinBatch : TextFile;
ArlequinProject : TextFile;
ScreenRatio : integer;
MuRatePerGen : extended;
MuRate : extended;
SeqLength : extended;
SeqLengthInt : integer;
NumberMutations : integer;
kSamples : integer;
MutLineages : array of array of integer;
MutLineages2 : array of array of integer;
pseudoNewick : string;
pseudoNewickArray : array of string;
sequence : array of char;
LineageClusters : array of array of string;
NucDiv, NucDivFin : real;
SegSites : integer;
TotalSegSitesFin : integer;
MuPerGene : extended;
FactorialResult, CombResult : extended;
xGamma : real;
NFactor : real;
StartN : integer;
NextN : integer;
NextNReal : real;
AddInd : real;
code : integer;
EndofRun : boolean;
locNumber, SizeLow, SizeHigh, repeatType : integer;
FileCounter : string;
ArlequinSampleSize : string;
valueChar : char;
NewickTreeFile : textfile;
Tmrca : array of integer;
TmrcaMSat : array of array of integer;
InfAlleles : array of integer;
InfAllelesMSat : array of array of integer;
```

```
  FinAlleles : array of integer;
  FinAllelesMSat : array of array of integer;
  GammaPerSiteAr : array of real;
  GammaGeneMuAr : array of real;
  AllelicDiversity : array of real;
  NucleotideDiversity : array of real;
  GeneDiversity : array of array of real;
  DirOut : string;
  TreeMatrixFile : textfile;
  MutationMatrixFile : textfile;
  TreeCounter : integer;
  SimDone : boolean;

const
  DevelopHeight = 1200;

procedure ChooseRandom(k, N : integer);
procedure SortRandom(Arrayk : array of integer; SampleSize : integer);
procedure MakeTree(Arrayk: array of integer; SampleSize: integer);
procedure Mutations(Samples: integer);
procedure NewickTree(Currentk, Originalk: integer);
procedure TrueNewickTree(Samples: integer);
procedure IdentifyLineageClusters(k : integer);
procedure MutateLineages(SeqLengthORLocus: integer);
procedure DiversityIndices(SeqLength, kSamples, SimNum: integer);
procedure MutationDistribution(Samples : integer);
procedure GeneMuRate(MuRatePerSitePerGen : extended; GeneLength: integer);
procedure MicrosatModel(Samples, PopSize, GStart, Gend, NSims : integer);
procedure GammaMutations(Samples,SeqLength, NumSims: integer;
alpha,MuRatePerGen: real);
procedure MicrosatMutations(Samples: integer; MicroRate : real);
procedure LogFile(kL,NL,StartNL,SeqLengthL,GrowthStartL,GrowthEndL,
NumberSimulationsL: integer; StartTimeL, EndTimeL : TDateTime;
SeqRateL,FreqAL,FreqCL,FreqGL,FreqTL : real);
procedure Outputs(NumberSims,NLoci,NumSamples: integer);
procedure DrawTree;

function JC69(CurrentBase: char): char;
function Kimura2Param(CurrentBase: char): char;
function Felsenstein81(CurrentBase: char): char;
function HKY85(CurrentBase: char): char;
function Factorial(Number: integer): extended;
function nChooseR (val1, val2 : integer) :extended;
function GammaMultiplication(a: real):real;
function GammaRejection(a: real): real;
function GammaTwoPart(a: real): real;
function MsatRandom(allele: integer): integer;
function MsatStepwise(allele: integer): integer;
```

```
implementation

{$R *.xfm}
{
procedure TfrmRandCoal.Scale(Sender: TObject);       //need to fix the resolution
problems, each button needs to be scaled accordingly

var
  ScreenHeight : integer;
  ScreenWidth : integer;

begin
  with Screen do
    ScreenHeight := Height;
    ScreenWidth := Width;
    ScreenRatio := DevelopHeight div ScreenHeight;
    ScaleBy(ScreenHeight,DevelopHeight);
    frmRandCoal.Height := ScreenHeight;
    frmRandCoal.Width := ScreenWidth;
end;
}
procedure ChooseRandom(k, N : integer);

//Chooses k Random numbers from N and assigns values to an array

var
  RandomK : integer;
  i : integer;

begin
  RandomkArray := nil;
  SetLength(RandomkArray, k);
  for i:=1 to k do begin
    Randomk := Random(N);
    RandomkArray[i-1] := RandomK;
  end;
end;

procedure SortRandom(Arrayk : array of integer; SampleSize : integer);

//Sorts the random values chosen in ChooseRandom

var
  sorted : boolean;
  l,i : integer;
  temp : integer;

begin
  repeat
```

```
    sorted := True;
    for l:=1 to SampleSize do begin
      if (Arrayk[l-1] > Arrayk[l]) then begin
        temp := Arrayk[l-1];
        Arrayk[l-1] := Arrayk[l];
        Arrayk[l] := temp;
        sorted:=False;
      end;
    end;
  until sorted;
  SortedRandArray := nil;
  setLength(SortedRandArray, SampleSize);
  for i := 1 to SampleSize do begin
    temp := Arrayk[i-1];
    SortedRandArray[i-1] := temp;
  end;
end;

procedure MakeTree(Arrayk: array of integer; SampleSize: integer);

//Identifies equal numbers in the k chosen from N, and assigns each identity
//event as a coalescent event. Also keeps track of the number of generations
//or the time to coalescence of each coalescent event. This procedure also
//writes the coalescent events with times and the descendents to a matrix called
//TreeMatrix.

var
  t : integer;
  TermA, TermB : integer;
  Diff : integer;
  CountCoalesceStart : integer;

begin
  CountCoalesceStart := CountCoalesce; //This code and the if loop ensures a max of
one coalescent
                                //...event per generation. You can allow polytomies by
removing it, but then it gets difficult to draw the tree and Newick format
    for t := 1 to SampleSize-1 do begin
      TermA := Arrayk[t-1];
      TermB := Arrayk[t];
      Diff := TermA - TermB;
      if (Diff = 0) and (CountCoalesce = CountCoalesceStart) then begin  //allow
polytomies by removing 2nd condition
        CountCoalesce := CountCoalesce+1;
        TreeMatrix[CountCoalesce-1, 1] := t-1;
        TreeMatrix[CountCoalesce-1, 2] := t;
        TreeMatrix[CountCoalesce-1, 0] := CoalesceTime;
      end;
    end;
end;
```

```
procedure GeneMuRate(MuRatePerSitePerGen : extended; GeneLength: integer);

begin
  MuPerGene := 1-Power((1-MuRatePerSitePerGen),GeneLength);
end;

function Factorial(Number: integer): extended;

begin
  FactorialResult := 1;
  If Number > 1 then begin
  repeat
    FactorialResult := FactorialResult*Number;
    Number := Number-1;
  until Number = 1;
  end;
end;

function nChooseR (val1, val2 : integer) :extended;

var
TermA, TermB, TermC : extended;

begin
  CombResult := 1;
  Factorial(val1);
  TermA := FactorialResult;
  Factorial(val2);
  TermB := FactorialResult;
  Factorial(val1-val2);
  TermC := FactorialResult;
  CombResult := TermA/((TermB)*(TermC));
end;

procedure Mutations(Samples: integer);

var
  RandomValue : extended;
  RandomLineage : integer;
  pMuPbPg : extended;
  pMuPbPgi : extended;
  CumPMuPbPgi : extended;
  i : integer;
  combin : extended;

begin
  RandomValue := Random;
  i := 1;
  pMuPbPg := 1-Power((1-MuPerGene), Samples);
```

```
  CumPMuPbPgi := 0;
  while ((PMuPbPg-CumPMuPbPgi) > RandomValue) do begin
    CountMutation := CountMutation+1;
    MutationMatrix[CountMutation-1, 0] := CoalesceTime;
    RandomLineage := Random(Samples);
    MutationMatrix[CountMutation-1, 1] := RandomLineage;
    nChooseR(Samples, i);
    Combin := CombResult;
    pMuPbPgi := (Power(MuPerGene,i)*Power((1-MuPerGene), Samples-i)*Combin);
    CumPMuPbPgi := CumPMuPbPgi+PMuPbPgi;
    i := i+1;
  end;
end;

function GammaMultiplication(a: real):real;
//the multiplication method for generating a gamma random variable when alpha
//is an integer. Ahrens & Dieter 1974 Computer methods for sampling from Gamma
//Beta, Poisson and Binomial distributions. Computing 12: 223-246

var
  j : integer;
  aInt : integer;
  RandomU : real;
  p : real;

begin
  aInt := Trunc(a);
  j := 1;
  p := j;
  RandomU := Random;
  p := p*RandomU;
  while (j <> aInt) do begin
    RandomU := Random;
    p := p*RandomU;
    j := j+1;
  end;
  xGamma := -ln(p);
end;

function GammaRejection(a: real): real;
//this function uses a rejection method to draw from a gamma distribution.
//Ahrens & Dieter 1974 Computer methods for sampling from Gamma
//Beta, Poisson and Binomial distributions. Computing 12: 223-246

var
  RandomU : real;
  RandomU2 : real;
  FuncB : real;
  FuncP : real;
```

```
begin
  repeat
    RandomU := random;
    FuncB := (exp(1) + a)/exp(1);
    FuncP := FuncB*RandomU;
    if FuncP > 1 then begin
      xGamma := -ln((FuncB-FuncP)/a);
      RandomU2 := random;
    end
    else begin
      xGamma := Power(FuncP, (1/a));
      RandomU2 := random;
    end;
  until (RandomU2 < exp(-xGamma)) or (RandomU2 < Power(xGamma, (a-1)));
end;

function GammaTwoPart(a: real): real;
//if alpha is real and > 1 then we use a two-part method to sample from the
//gamma distribution. this method breaks the real number into an integer part, and
//a real part and uses the methods above in combination.
//Ahrens & Dieter 1974 Computer methods for sampling from Gamma
//Beta, Poisson and Binomial distributions. Computing 12: 223-246

var
  aInt : integer;
  j : integer;
  p : real;
  RandomU : real;
  RandomU2 : real;
  xGammaTwoPart1 : real;
  xGammaTwoPart2 : real;
  FuncB, FuncP : real;

begin
  aInt := Trunc(a);
  j := 1;
  p := j;
  RandomU := Random;
  p := p*RandomU;
  while (j <> aInt) do begin
    RandomU := Random;
    p := p*RandomU;
    j := j+1;
  end;
  xGammaTwoPart1 := -ln(p);
  repeat
    RandomU := random;
    FuncB := (exp(1) + a)/exp(1);
    FuncP := FuncB*RandomU;
    if FuncP > 1 then begin
```

```
      xGammaTwoPart2 := -ln((FuncB-FuncP)/a);
      RandomU2 := random;
    end
    else begin
      xGammaTwoPart2 := Power(FuncP, (1/a));
      RandomU2 := random;
    end;
  until (RandomU2 < exp(-xGammaTwoPart2)) or (RandomU2 <
Power(xGammaTwoPart2, (a-1)));
  xGamma := xGammaTwoPart1+xGammaTwoPart2;
end;

procedure GammaMutations(Samples,SeqLength,NumSims : integer;
alpha,MuRatePerGen: real);

var
  RandomValue : extended;
  RandomLineage : integer;
  pMuPbPg : extended;
  pMuPbPgi : extended;
  CumPMuPbPgi : extended;
  i : integer;
  combin : extended;
  testAlpha : real;
  MuPerGenG : real;
  GammaGeneMu : real;
  StringG : string;

begin

  testAlpha := alpha-Trunc(alpha);
  if (testAlpha = 0) then  //i.e. is alpha an integer
    GammaMultiplication(alpha)
  else if (testAlpha <> 0) and (alpha > 0) and (alpha < 1) then //i.e. 0 < alpha < 1
    GammaRejection(alpha)
  else if (testAlpha <> 0) then
    GammaTwoPart(alpha); //i.e alpha is real and > 1

  MuPerGenG := (XGamma/alpha)*MuRatePerGen;
  GammaPerSiteAr[NumSims-1] := MuPerGenG;
  GammaGeneMu := 1-Power((1-MuPerGenG),SeqLength);
  GammaGeneMuAr[NumSims-1] := GammaGeneMu;

  RandomValue := Random;
  i := 1;
  pMuPbPg := 1-Power((1-MuPerGene), Samples);
  CumPMuPbPgi := 0;
  while ((PMuPbPg-CumPMuPbPgi) > RandomValue) do begin
    CountMutation := CountMutation+1;
    MutationMatrix[CountMutation-1, 0] := CoalesceTime;
```

```
    RandomLineage := Random(Samples);
    MutationMatrix[CountMutation-1, 1] := RandomLineage;
    nChooseR(Samples, i);
    Combin := CombResult;
    pMuPbPgi := (Power(MuPerGene,i)*Power((1-MuPerGene), Samples-i)*Combin);
    CumPMuPbPgi := CumPMuPbPgi+PMuPbPgi;
    i := i+1;
  end;
end;

procedure MicrosatMutations(Samples: integer; MicroRate : real);

var
  RandomValue : extended;
  RandomLineage : integer;
  pMuPbPg : extended;
  pMuPbPgi : extended;
  CumPMuPbPgi : extended;
  i : integer;
  combin : extended;

begin
  RandomValue := Random;
  i := 1;
  pMuPbPg := 1-Power((1-MicroRate), Samples);
  CumPMuPbPgi := 0;
  while ((PMuPbPg-CumPMuPbPgi) > RandomValue) do begin
    CountMutation := CountMutation+1;
    MutationMatrix[CountMutation-1, 0] := CoalesceTime;
    RandomLineage := Random(Samples);
    MutationMatrix[CountMutation-1, 1] := RandomLineage;
    nChooseR(Samples, i);
    Combin := CombResult;
    pMuPbPgi := (Power(MicroRate,i)*Power((1-MicroRate), Samples-i)*Combin);
    CumPMuPbPgi := CumPMuPbPgi+PMuPbPgi;
    i := i+1;
  end;
end;

procedure NewickTree(Currentk, Originalk: integer);

{Please note that the structure of this tree is not completely in Newick format. The
coalescent
events are correct but the branch lengths are not. The purpose of the branch lengths in
this
tree is to allow us to determine which samples cluster at which times such that we can
identify
the correct lineages to mutate when a mutation occurs. Later in the program, when the
TreeMatrix
is complete, I assemble a true Newick format tree}
```

```
var
  CoalEvents : integer;
  SampleA, SampleB : integer;
  NewickA, NewickB : string;
  BrLengthInt : integer;
  BrLength : string;
  i,j : integer;
  NewickCell : string;


begin
  CoalEvents := Originalk-Currentk;
  If (CoalEvents = 1) then begin
    for i:=0 to Originalk-1 do begin
      NewickArray[CoalEvents-1, i] := IntToStr(i);
    end;
  end;
  BrLengthInt := TreeMatrix[CoalEvents-1, 0];
  BrLength := IntToStr(BrLengthInt);
  SampleA := TreeMatrix[CoalEvents-1, 1];
  SampleB := TreeMatrix[CoalEvents-1, 2];
  NewickA := NewickArray[CoalEvents-1, SampleA];
  NewickB := NewickArray[CoalEvents-1, SampleA+1];
  j := 0;
  while (j < Originalk) do begin
    NewickArray[CoalEvents, j] := NewickArray[CoalEvents-1, j];;
    If (j = SampleA) then begin
      NewickCell := '('+NewickA+','+NewickB+'):'+BrLength;
      NewickArray[CoalEvents, j] := NewickCell;
    end;
    If (j >= SampleB) and (j <> Originalk-1) then begin
      NewickCell := NewickArray[CoalEvents-1, j+1];
      NewickArray[CoalEvents, j] := NewickCell;
    end;
    j := j+1;
  end;
end;

procedure TrueNewickTree(Samples: integer);

var
  i,j,l,m,n,o : integer;
  q,t,u,v,w,x : integer;
  SampleA, SampleB : integer;
  NewickA, NewickB : string;
  Branch1 : integer;
  Branch2A, Branch2B : integer;
  BranchDiffA, BranchDiffB : integer;
  TestChar, TestChar2 : char;
```

```
  ParenthesesA, ParenthesesB : boolean;
  BranchStr : string;
  NewickCell : string;
  BranchInt,BranchAdd : integer;
  BranchTest2 : integer;
  Count40_A, Count40_B : integer;
  Count40Test, Count41Test : integer;


begin
  NewickArray2 := nil;
  SetLength(NewickArray2, Samples, Samples);

  for i := 0 to Samples-1 do begin
    NewickArray2[0,i] := IntToStr(i);
  end;

  for j:= 0 to Samples-2 do begin
    SampleA := TreeMatrix[j, 1];
    SampleB := TreeMatrix[j, 2];
    NewickA := NewickArray2[j, SampleA];
    NewickB := NewickArray2[j, SampleA+1];
    Branch1 := TreeMatrix[j,0];

    l := 0;
    while (l < Samples) do begin
      NewickArray2[j+1, l] := NewickArray2[j, l];;
      If (l = SampleA) then begin

        //are there parentheses in NewickA or NewickB
        ParenthesesA := False;
        ParenthesesB := False;
        m := 1;
        repeat
          TestChar := NewickA[m];
          If (Ord(TestChar) = 40) then
            ParenthesesA := True;
          m := m+1;
        until (Ord(TestChar) = 40) or (m >= Length(NewickA));

        n := 1;
        repeat
          TestChar := NewickB[n];
          If (Ord(TestChar) = 40) then
            ParenthesesB := True;
          n := n+1;
        until (Ord(TestChar) = 40) or (n >= Length(NewickB));

        If (ParenthesesA = True) and (ParenthesesB = False) then begin
          BranchAdd := 0;
```

```
        q := Length(NewickA);
        repeat
          TestChar := NewickA[q];
          SetLength(BranchStr, 0);
          if (Ord(TestChar) = 58) then begin
            o := q+1;
            repeat
              TestChar := NewickA[o];
              BranchStr := BranchStr+TestChar;
              o := o+1;
            until (Ord(TestChar)=44) or (Ord(TestChar)=40) or (Ord(TestChar)=41);
            SetLength(BranchStr, Length(BranchStr)-1);
            BranchAdd := BranchAdd + StrToInt(BranchStr);
          end;
          q := q-1;
        until (Ord(TestChar) = 44);

        BranchDiffA := Branch1-(BranchAdd);
        NewickCell :=
'('+NewickA+':'+IntToStr(BranchDiffA)+','+NewickB+':'+IntToStr(Branch1)+')';
        NewickArray2[j+1, l] := NewickCell;
      end

      else If (ParenthesesA = False) and (ParenthesesB = True) then begin
        BranchAdd := 0;
        q := Length(NewickB);
        repeat
          TestChar := NewickB[q];
          SetLength(BranchStr, 0);
          if (Ord(TestChar) = 58) then begin
            o := q+1;
            repeat
              TestChar := NewickB[o];
              BranchStr := BranchStr+TestChar;
              o := o+1;
            until (Ord(TestChar)=44) or (Ord(TestChar)=40) or (Ord(TestChar)=41);
            SetLength(BranchStr, Length(BranchStr)-1);
            BranchAdd := BranchAdd + StrToInt(BranchStr);
          end;
          q := q-1;
        until (Ord(TestChar) = 44);

        BranchDiffB := Branch1-(BranchAdd);
        NewickCell :=
'('+NewickA+':'+IntToStr(Branch1)+','+NewickB+':'+IntToStr(BranchDiffB)+')';
        NewickArray2[j+1, l] := NewickCell;
      end

      else if (ParenthesesA = True) and (ParenthesesB = True) then begin
```

```
        BranchAdd := 0;

        q := Length(NewickA);
        repeat
          TestChar := NewickA[q];
          SetLength(BranchStr, 0);
          if (Ord(TestChar) = 58) then begin
            o := q+1;
            repeat
              TestChar := NewickA[o];
              BranchStr := BranchStr+TestChar;
              o := o+1;
            until (Ord(TestChar)=44) or (Ord(TestChar)=40) or (Ord(TestChar)=41);
            SetLength(BranchStr, Length(BranchStr)-1);
            BranchAdd := BranchAdd + StrToInt(BranchStr);
          end;
          q := q-1;
        until (Ord(TestChar) = 44);

        BranchDiffA := Branch1-(BranchAdd);

        BranchAdd := 0;
        q := Length(NewickB);
        repeat
          TestChar := NewickB[q];
          SetLength(BranchStr, 0);
          if (Ord(TestChar) = 58) then begin
            o := q+1;
            repeat
              TestChar := NewickB[o];
              BranchStr := BranchStr+TestChar;
              o := o+1;
            until (Ord(TestChar)=44) or (Ord(TestChar)=40) or (Ord(TestChar)=41);
            SetLength(BranchStr, Length(BranchStr)-1);
            BranchAdd := BranchAdd + StrToInt(BranchStr);
          end;
          q := q-1;
        until (Ord(TestChar) = 44);

        BranchDiffB := Branch1-(BranchAdd);

        NewickCell :=
'('+NewickA+':'+IntToStr(BranchDiffA)+','+NewickB+':'+IntToStr(BranchDiffB)+')';
        NewickArray2[j+1, l] := NewickCell;
      end

      else begin
        NewickCell :=
'('+NewickA+':'+IntToStr(Branch1)+','+NewickB+':'+IntToStr(Branch1)+')';
        NewickArray2[j+1, l] := NewickCell;
```

```
      end;
    end;

    If (l >= SampleB) and (l <> Samples-1) then begin
      NewickCell := NewickArray2[j, l+1];
      NewickArray2[j+1, l] := NewickCell;
    end;
    l := l+1;
  end;
 end;
end;

procedure IdentifyLineageClusters(k : integer);

var
 i,j,m,l : integer;
 clusterTime : integer;
 outfile : textfile;
 TextString : string;

begin
 LineageClusters := nil;
 SetLength(LineageClusters, k-1, k+1);
 for i:=0 to High(TreeMatrix) do begin
  clusterTime := TreeMatrix[i, 0];
  LineageClusters[i, 0] := IntToStr(clusterTime);
  for j:=1 to k-1 do begin
   LineageClusters[i,j] := NewickArray[i+1,j-1];
  end;
 end;
end;

procedure MutateLineages(SeqLengthORLocus : integer);

var
 MutationTime : integer;
 MutLineage, MutLineage2 : integer;
 ii,jj,nn,mm : integer;
 CurrentBase : char;
 RandomSite : integer;
 clusterTime : integer;
 clusterTimeStr : string;
 LineageString : string;
 CharN : char;
 CharNMinus : char;
 CharNPlus : char;
 result : char;
 resultm : integer;
 StringN : string;
 Count : integer;
```

```
  RandomAllele : real;
  RandomAlleleInt : integer;
  CurrentAllele : integer;

begin
  LineageString := 'd';
  StringN := 'k';
  MutLineage2 := 0;

  for ii:=CountMutation-1 downto 0 do begin     //Starting at the last mutation in the
genealogy...

    //the ff code identifies the time and lineage of the iith mutation. The iith lineage in
this case is one
    //of the (k-number Coalescent events) lineages in the current waiting time.
Therefore, we need to identify
    //the lineage clusters at each mutation time, and assign the cluster of lineages
mutated by the mutation
    //to a string LineageString.

    MutationTime := MutationMatrix[ii,0];
    MutLineage := MutationMatrix[ii,1];
    jj := 0;
    clusterTimeStr := LineageClusters[jj,0];
    clusterTime := StrToInt(clusterTimeStr);
    while (MutationTime > clusterTime) do begin
      jj := jj+1;
      clusterTimeStr := LineageClusters[jj,0];
      clusterTime := StrToInt(clusterTimeStr);
    end;

    If (jj = 0) then begin   //i.e. mutation occurs before coalescent events and therefore
MutLineage is the Sample
                          //number and only one sample should be mutated
      LineageString := IntToStr(MutLineage);
    end
    else begin
    LineageString := LineageClusters[jj-1, MutLineage+1];
    end;

    //now we find the current base/allele of the first sample in the lineage cluster to be
mutated

    nn := 1;
    CharN := LineageString[nn];
    Count := 0;   //since we only need the current base of the first lineage to input to the
mutation model
    while (Count < 1) and (nn <= Length(LineageString)) do begin
```

```
    if (Ord(CharN) <> 40) and (Ord(CharN) <> 41) and (Ord(CharN) <> 44) and
(Ord(CharN) <> 58) then begin //if char is not a ( ) , or : then it must be a lineage or
coalescent time
      CharNMinus := LineageString[nn-1];
      if (Ord(CharNMinus) <> 58) and (nn < Length(LineageString)) then begin
//char is not a coalescent time
       CharNPlus := LineageString[nn+1];   //char following is what?
       StringN := CharN;
       while (Ord(CharNPlus) <> 40) and (Ord(CharNPlus) <> 41) and
(Ord(CharNPlus) <> 44) and (Ord(CharNPlus) <> 58) and (nn <
Length(LineageString)) do begin
         StringN := StringN+CharNPlus;    //if ff char is not (),: then it is a number that
should be collated with the sample eg. ( 3 4 , = 34
         nn := nn+1;
         CharNPlus := LineageString[nn+1];
       end;
       MutLineage2 := StrToInt(StringN);
       Count := Count+1;
      end
      else if (Ord(CharNMinus) = 58) then begin     //if preceding char is a : then the
number is a coalescent time
       CharNPlus := LineageString[nn+1];
       while (Ord(CharNPlus) <> 44) and (Ord(CharNPlus) <> 40) and
(Ord(CharNPlus) <> 41) do begin //or part of a coalescent time
         nn := nn+1;
         CharNPlus := LineageString[nn+1];
       end;
      end;
    end;
    nn := nn+1;
    CharN := LineageString[nn];
  end;

  //now we determine the mutation that occurs from current base to ? at a random site
  //under the selected mutation model for sequences or microsatellite model

  with frmRandCoal do begin
   if chkMsats.Checked then begin
    CurrentAllele := MData[MutLineage2, SeqLengthORLocus];
    if radAllele.Checked then
      resultm := MsatRandom(CurrentAllele);
    if radStepwise.Checked then
      resultm := MSatStepwise(CurrentAllele);
   end
   else begin
    RandomSite := Random(SeqLengthORLocus);  //between 0 and 999 which is
fine since Data is an array with index 0>999
    CurrentBase := Data[RandomSite, MutLineage2];
    If radJC69.checked then
      result := JC69(CurrentBase);
```

```
      If radK2P.checked then
        result := Kimura2Param(CurrentBase);
      If radF81.checked then
        result := Felsenstein81(CurrentBase);
      If radHKY85.checked then
        result := HKY85(CurrentBase);
    end;
  end;


  //now we enforce the above mutations on the samples in the current lineage cluster

  mm := 1;
  repeat

    CharN := LineageString[mm];
    if (Ord(CharN) <> 40) and (Ord(CharN) <> 41) and (Ord(CharN) <> 44) and
(Ord(CharN) <> 58) then begin
      CharNMinus := LineageString[mm-1];

      if (Ord(CharNMinus) <> 58) and (mm <> Length(LineageString)) then begin
        CharNPlus := LineageString[mm+1];
        StringN := CharN;
        while (Ord(CharNPlus) <> 40) and (Ord(CharNPlus) <> 41) and
(Ord(CharNPlus) <> 44) and (Ord(CharNPlus) <> 58) and (mm <>
Length(LineageString)) do begin
          StringN := StringN+CharNPlus;
          mm := mm+1;
          CharNPlus := LineageString[mm+1];
        end;
        MutLineage2 := StrToInt(StringN);
        with frmRandCoal do begin
          if chkMsats.Checked then begin
            MData[MutLineage2, SeqLengthORLocus] := resultm;
          end
          else begin
            Data[RandomSite, MutLineage2] := result;
          end;
        end;
      end;

      if (Ord(CharNMinus) = 58) then begin
        CharNPlus := LineageString[mm+1];
        while (Ord(CharNPlus) <> 44) and (Ord(CharNPlus) <> 40) and
(Ord(CharNPlus) <> 41) and (mm < Length(LineageString)) do begin
          mm := mm+1;
          CharNPlus := LineageString[mm+1];
        end;
      end;
```

```
      end;
      mm := mm+1;

    until (mm >= Length(LineageString));

  end;
end;



function JC69(CurrentBase: char): char;

var
  RandomMutation : integer;

begin
  case Ord(CurrentBase) of
    65, 97 : begin     //a, A
               RandomMutation := Random(3);
               case RandomMutation of
                 0 : begin
                       result := 'c';
                     end;
                 1 : begin
                       result := 'g';
                     end;
                 2 : begin
                       result := 't';
                     end;
               end;
             end;
    67, 99 : begin     //c, C
               RandomMutation := Random(3);
               case RandomMutation of
                 0 : begin
                       result := 'a';
                     end;
                 1 : begin
                       result := 'g';
                     end;
                 2 : begin
                       result := 't';
                     end;
               end;
             end;
    71,103 : begin     //g, G
               RandomMutation := Random(3);
               case RandomMutation of
                 0 : begin
                       result := 'a';
```

```
              end;
            1 : begin
                result := 'c';
              end;
            2 : begin
                result := 't';
              end;
          end;
        end;
  84,116 : begin     //t, T
          RandomMutation := Random(3);
          case RandomMutation of
            0 : begin
                result := 'a';
              end;
            1 : begin
                result := 'c';
              end;
            2 : begin
                result := 'g';
              end;
          end;
        end;
 end;
end;

function Kimura2Param(CurrentBase: char): char;

var
 RandomMutation : integer;
 TiTvRatio : real;
 code : integer;
 RandomAdd : real;
 RAI : integer;

begin
 with frmRandCoal do begin
  Val(edtTiTv.Text, TiTvRatio, code);
 end;
 RandomAdd := TiTvRatio*100;   //TiTv ratio must be to 2 decimal places or less
 RAI := Trunc(RandomAdd);

 case Ord(CurrentBase) of
  65, 97 : begin     //a, A
          RandomMutation := Random(200+RAI);
          if (RandomMutation >=0) and (RandomMutation < RAI) then
            result := 'g'     //Ti
          else if (RandomMutation >= RAI) and (RandomMutation < RAI+100) then
            result := 'c'     //Tv
```

```
            else if (RandomMutation >= RAI+100) and (RandomMutation < RAI+200)
then
              result := 't';     //Tv
            end;
    67, 99 : begin     //c, C
            RandomMutation := Random(200+RAI);
            if (RandomMutation >=0) and (RandomMutation < RAI) then
              result := 't'     //Ti
            else if (RandomMutation >= RAI) and (RandomMutation < RAI+100) then
              result := 'a'     //Tv
            else if (RandomMutation >= RAI+100) and (RandomMutation < RAI+200)
then
              result := 'g';     //Tv
            end;
    71,103 : begin     //g, G
            RandomMutation := Random(200+RAI);
            if (RandomMutation >=0) and (RandomMutation < RAI) then
              result := 'a'     //Ti
            else if (RandomMutation >= RAI) and (RandomMutation < RAI+100) then
              result := 'c'     //Tv
            else if (RandomMutation >= RAI+100) and (RandomMutation < RAI+200)
then
              result := 't';     //Tv
            end;
    84,116 : begin     //t, T
            RandomMutation := Random(200+RAI);
            if (RandomMutation >=0) and (RandomMutation < RAI) then
              result := 'c'     //Ti
            else if (RandomMutation >= RAI) and (RandomMutation < RAI+100) then
              result := 'a'     //Tv
            else if (RandomMutation >= RAI+100) and (RandomMutation < RAI+200)
then
              result := 'g';     //Tv
            end;
  end;
end;


function Felsenstein81(CurrentBase: char): char;

var
  FreqA, FreqC, FreqG, FreqT : real;
  code : integer;
  RandomMutation : integer;
  RandomFrom : real;
  RFI : integer;

begin
  with frmRandCoal do begin
    Val(edtFreqA.Text, FreqA, code);
```

```
  Val(edtFreqC.Text, FreqC, code);
  Val(edtFreqG.Text, FreqG, code);
 end;

 FreqT := 1-(FreqA+FreqC+FreqG);

 case Ord(CurrentBase) of
  65, 97 : begin     //a, A
            RandomFrom := (FreqC+FreqG+FreqT)*100;
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
            if (RandomMutation >=0) and (RandomMutation < FreqC*100) then
              result := 'c'
            else if (RandomMutation >= FreqC*100) and (RandomMutation <
(FreqC+FreqG)*100) then
              result := 'g'
            else if (RandomMutation >= (FreqC+FreqG)*100) and (RandomMutation <
(FreqC+FreqG+FreqT)*100) then
              result := 't';
           end;
  67, 99 : begin     //c, C
            RandomFrom := (FreqA+FreqG+FreqT)*100;
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
            if (RandomMutation >=0) and (RandomMutation < FreqA*100) then
              result := 'a'
            else if (RandomMutation >= FreqA*100) and (RandomMutation <
(FreqA+FreqG)*100) then
              result := 'g'
            else if (RandomMutation >= (FreqA+FreqG)*100) and (RandomMutation <
(FreqA+FreqG+FreqT)*100) then
              result := 't';
           end;
  71,103 : begin     //g, G
            RandomFrom := (FreqA+FreqC+FreqT)*100;
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
            if (RandomMutation >=0) and (RandomMutation < FreqA*100) then
              result := 'a'
            else if (RandomMutation >= FreqA*100) and (RandomMutation <
(FreqA+FreqC)*100) then
              result := 'c'
            else if (RandomMutation >= (FreqA+FreqC)*100) and (RandomMutation <
(FreqA+FreqC+FreqT)*100) then
              result := 't';
           end;
  84,116 : begin     //t, T
            RandomFrom := (FreqA+FreqC+FreqG)*100;
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
```

```
            if (RandomMutation >=0) and (RandomMutation < FreqA*100) then
              result := 'a'
            else if (RandomMutation >= FreqA*100) and (RandomMutation <
(FreqA+FreqC)*100) then
              result := 'c'
            else if (RandomMutation >= (FreqA+FreqC)*100) and (RandomMutation <
(FreqA+FreqC+FreqG)*100) then
              result := 'g';
          end;
  end;

end;

function HKY85(CurrentBase: char): char;

var
  FreqA, FreqC, FreqG, FreqT : real;
  code : integer;
  RandomMutation : integer;
  TiTvRatio : real;
  RandomFrom : real;
  RFI : integer;


begin
  with frmRandCoal do begin
    Val(edtFreqA.Text, FreqA, code);
    Val(edtFreqC.Text, FreqC, code);
    Val(edtFreqG.Text, FreqG, code);
    Val(edtTiTv.Text, TiTvRatio, code);
  end;
  FreqT := 1-(FreqA+FreqC+FreqG);

  case Ord(CurrentBase) of
    65, 97 : begin     //a, A
            RandomFrom := (TiTvRatio*100*freqG)+(100*freqC)+(100*freqT);
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
            if (RandomMutation >=0) and (RandomMutation <
(TiTvRatio*100*freqG)) then
              result := 'g'     //Ti
            else if (RandomMutation >= TiTvRatio*100*freqG) and (RandomMutation
< ((TiTvRatio*100*freqG)+(100*freqC))) then
              result := 'c'     //Tv
            else if (RandomMutation >= ((TiTvRatio*100*freqG)+(100*freqC))) and
(RandomMutation < (TiTvRatio*100*freqG)+(100*freqC)+(100*freqT)) then
              result := 't';     //Tv
          end;
    67, 99 : begin     //c, C
            RandomFrom := (TiTvRatio*100*freqT)+(100*freqA)+(100*freqG);
```

```
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
            if (RandomMutation >=0) and (RandomMutation < (TiTvRatio*100*freqT))
then
              result := 't'     //Ti
            else if (RandomMutation >= TiTvRatio*100*freqT) and (RandomMutation
< ((TiTvRatio*100*freqT)+(100*freqA))) then
              result := 'a'     //Tv
            else if (RandomMutation >= ((TiTvRatio*100*freqT)+(100*freqA))) and
(RandomMutation < (TiTvRatio*100*freqT)+(100*freqA)+(100*freqG)) then
              result := 'g';     //Tv
          end;
   71,103 : begin     //g, G
           RandomFrom := (TiTvRatio*100*freqA)+(100*freqC)+(100*freqT);
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
            if (RandomMutation >=0) and (RandomMutation <
(TiTvRatio*100*freqA)) then
              result := 'a'     //Ti
            else if (RandomMutation >= TiTvRatio*100*freqA) and (RandomMutation
< ((TiTvRatio*100*freqA)+(100*freqC))) then
              result := 'c'     //Tv
            else if (RandomMutation >= ((TiTvRatio*100*freqA)+(100*freqC))) and
(RandomMutation < (TiTvRatio*100*freqA)+(100*freqC)+(100*freqT)) then
              result := 't';     //Tv
          end;
   84,116 : begin     //t, T
            RandomFrom := (TiTvRatio*100*freqC)+(100*freqA)+(100*freqG);
            RFI := Trunc(RandomFrom);
            RandomMutation := Random(RFI);
            if (RandomMutation >=0) and (RandomMutation < (TiTvRatio*100*freqC))
then
              result := 'c'     //Ti
            else if (RandomMutation >= TiTvRatio*100*freqC) and (RandomMutation
< ((TiTvRatio*100*freqC)+(100*freqA))) then
              result := 'a'     //Tv
            else if (RandomMutation >= ((TiTvRatio*100*freqC)+(100*freqA))) and
(RandomMutation < (TiTvRatio*100*freqC)+(100*freqA)+(100*freqG)) then
              result := 'g';     //Tv
          end;
  end;
end;

function MsatRandom(allele: integer): integer;   //choose a random allele within the
size range

var
  SizeDiff : integer;
  NewAllele : integer;
```

```
begin
  SizeDiff := SizeHigh-SizeLow;
  repeat
    NewAllele := Trunc(Random(SizeDiff));
  until (NewAllele mod repeatType = 0);
  result := NewAllele+SizeLow;
end;

function MsatStepwise(allele: integer): integer;  //choose an allele one step size
smaller or larger than the current

var
  NewAllele : integer;

begin
  if (Random > 0.5) then
    NewAllele := allele+repeatType
  else
    NewAllele := allele-repeatType;
  result := NewAllele;
end;

procedure MutationDistribution(Samples : integer);

var
  i,j,k,l,m: integer;
  MutTime : integer;
  CoalTime : integer;
  DiffTime : integer;
  TreeLength : array of array of integer;
  BranchLengths : array of array of integer;
  TotalTreeLength : integer;
  MuTreeLength : integer;
  outfile : textfile;
  value : integer;
  result : string;
  BranchLength : integer;
  MutDistrib : array of array of integer;


begin
  TotalTreeLength := 0;
  TreeLength := nil;
  BranchLengths := nil;
  SetLength(TreeLength, Samples-1, 2);
  SetLength(BranchLengths, Samples-1, 2);
  for i:=0 to High(TreeMatrix) do begin
    TreeLength[i,0] := TreeMatrix[i,0];
    if (i=0) then begin
      TreeLength[i,1] := TreeMatrix[i,0]*(Samples);
```

```
    end
    else begin
      TreeLength[i,1] := (TreeMatrix[i,0]-TreeMatrix[i-1,0])*(Samples-
i)+TreeLength[i-1,1];
      BranchLengths[i,1] := (TreeMatrix[i,0]-TreeMatrix[i-1,0])*(Samples-i);
    end;
  end;

  //check code for tree length calculations

  Append(outfileCoalDis);
  Writeln(outfileCoalDis, 'Time of Coalescence, Total genealogy length');
  for j := 0 to High(TreeLength) do begin
    value := TreeLength[j,0];
    Str(value, result);
    Write(outfileCoalDis, result);
    Write(outfileCoalDis, ',');
    value := TreeLength[j,1];
    Str(value, result);
    Write(outfileCoalDis, result);
    Writeln(outfileCoalDis);
  end;
  CloseFile(outfileCoalDis);

  MutDistrib := nil;
  setLength(MutDistrib, CountMutation, 2);

  for k:=0 to CountMutation-1 do begin
    MuTreeLength := 0;
    MutTime := MutationMatrix[k,0];
    l := 0;
    CoalTime := TreeMatrix[l,0];

    //mutation occurs before any coalescent events
    if (MutTime < CoalTime) then begin
      DiffTime := CoalTime-MutTime;
      MuTreeLength := DiffTime*Samples;
    end

    //mutation occurs after coalescent events and treelength is calculated up to
    //when coalescent time > mutime
    else begin
      while (CoalTime < MutTime) do begin
        BranchLength := BranchLengths[l,1];
        MuTreeLength := MuTreeLength + BranchLength;
        l := l+1;
        CoalTime := TreeMatrix[l,0];
      end;
```

```
    //when mutation occurs between coalescent events the difference is
added*numberSamples
      if (MutTime < CoalTime) then begin
        DiffTime := MutTime-TreeMatrix[l-1,0];
        MuTreeLength := MuTreeLength+(DiffTime*(Samples-l));
      end;
    end;

    MutDistrib[k,0] := MutTime;
    MutDistrib[k,1] := MuTreeLength;

  end;

  Append(outfileMuDis);
  Writeln(outfileMuDis, 'Time of mutation, Total tree length');
  for m:=0 to High(MutDistrib) do begin

    value := MutDistrib[m,0];
    str(value, result);
    Write(outfileMuDis, result);
    Write(outfileMuDis, ',');

    value := MutDistrib[m,1];
    str(value, result);
    Write(outfileMuDis, result);

    Writeln(outfileMuDis);
  end;
  CloseFile(outfileMuDis);
end;

procedure DiversityIndices(SeqLength, kSamples, SimNum : integer);

var
  i,j,l : integer;
  m,n,o : integer;
  Base1, Base2 : char;
  SegSitesFin : integer;
  Sequence1, Sequence2 : integer;
  Base1U, Base2U : char;
  CountDifferences : integer;
  PairwiseDist : real;
  PairwiseDistArray : array of real;
  CountZero : integer;
  Base1Str, Base2Str : string;
  SumPairwise : real;

begin
  SegSites := 0;
  with frmRandCoal do begin
```

```
if chkSegSites.checked then begin
  SegSites := CountMutation;
end;
if chkScatterMut.checked then begin
  for m:=0 to SeqLength-1 do begin
    n := 0;
    SegSitesFin := 0;
    while (SegSitesFin <> 1) and (n <= kSamples-2) do begin
      Base1 := Data[m,0];
      Base2 := Data[m, n+1];
      if (Ord(Base1) <> Ord(Base2)) then begin
        SegSitesFin := SegSitesFin+1;
      end;
      n := n+1;
    end;
    TotalSegSitesFin := TotalSegSitesFin+SegSitesFin;
  end;
end;

if chkSeqDiversity.checked then begin
  l := 0;
  CountZero := 0;
  PairwiseDistArray := nil;
  SetLength(PairwiseDistArray, ((kSamples*(kSamples-1)) div 2));
  for i := 1 to kSamples-1 do begin
    Sequence1 := i;
    Sequence2 := i;
    repeat
      CountDifferences := 0;
      Sequence2 := Sequence2+1;
      for j:=1 to SeqLength do begin
        Base1 := Data[j-1,Sequence1-1];
        Base2 := Data[j-1,Sequence2-1];
        Base1Str := UpperCase(Base1);
        Base2Str := UpperCase(Base2);
        Base1 := Base1Str[1];
        Base2 := Base2Str[1];
        if (Ord(Base1) <> Ord(Base2)) then
          CountDifferences := CountDifferences+1;
      end;
      PairwiseDist := CountDifferences/SeqLength;
      if (PairwiseDist > 0) then begin
        PairwiseDistArray[l] := PairwiseDist;
        l := l+1;
      end;
      if (PairwiseDist = 0) then begin
        CountZero := CountZero+1;
      end;
    until (Sequence2 = kSamples);
  end;
```

```
      AllelicDiversity[SimNum-1] := (((kSamples*(kSamples-1))/2-
CountZero))/((kSamples*(kSamples-1))/2);
      o := 0;
      repeat
        PairwiseDist := PairwiseDistArray[o];
        SumPairWise := SumPairWise+PairwiseDist;
        o := o+1;
      until (PairwiseDist = 0);
      NucleotideDiversity[SimNum-1] := (2/(kSamples*(kSamples-1)))*SumPairWise;
    end;
  end;
end;

procedure MicrosatModel(Samples, PopSize, GStart, GEnd, NSims : integer);

var
  i,j,q,s,t,a,b : integer;
  ri,rj,rk,rl,rm,rn,ro,rp,rq,rt,rr,rs : integer;
  gd : integer;
  ai, aj, ak : integer;
  MDataTemp, MDataTemp2 : array of array of integer;
  SortedMData : array of array of integer;
  AlleleFrequency : array of real;
  tempInt, tempInt2 : integer;
  CountAlleles : integer;
  NumberAlleles : array of integer;
  value2, value3 : integer;
  MsatInput : Textfile;
  NumLoci : integer;
  MSatMuRate : real;
  StartAllele1, StartAllele2 : integer;
  RangeSize : integer;
  RandomAllele : real;
  IntRandomAllele : integer;
  MDataOut : TextFile;
  CurrentAllele : integer;
  AlleleInt : integer;
  AlleleStr : string;
  MAlleles, MAlleles2 : TextFile;
  value : integer;
  result : string;
  sorted : boolean;
  StrValue : string;
  RandomChosenM : array of integer;
  ts, tt : integer;
  RandomMSat : real;
  RandomMSatInt : integer;
  AlreadyUsed : boolean;
  tempRandom : integer;
  individ : integer;
```

```
  Allele1 : integer;
  StringName : string;
  AlleleCount : integer;
  AlleleFreq : real;
  SumSquares : real;
  tmf,tmf2 : integer;
  mmf,mmf2 : integer;
  NumberSoFar : integer;

begin
  SampleSize := Samples;

  with frmRandCoal do begin
    AssignFile(MsatInput, DirOut + '/' + edtMsatIn.Text);
    Val(edtNumLoci.Text, NumLoci, code);
    AssignFile(NewickTreeFile, DirOut + '/' + edtNewickTrees.Text);
    Rewrite(NewickTreeFile);
    if chkMSatOut.checked then begin
      AssignFile(MDataOut, DirOut + '/' + edtMSatData.Text);
      Rewrite(MDataOut);
    end;
  end;

  Tmrca := nil;
  InfAllelesMSat := nil;
  FinAllelesMSat := nil;
  GeneDiversity := nil;
  SetLength(TmrcaMsat, NSims, NumLoci);
  SetLength(InfAllelesMsat, NSims, NumLoci);
  SetLength(FinAllelesMSat, NSims, NumLoci);
  SetLength(GeneDiversity, NSims, NumLoci);

  AssignFile(TreeMatrixFile, DirOut + '/' + 'TreeMatFile.txt');
  Rewrite(TreeMatrixFile);
  AssignFile(MutationMatrixFile, DirOut + '/' + 'MutMatFile.txt');
  Rewrite(MutationMatrixFile);
  NumberMutationArray := nil;
  SetLength(NumberMutationArray, NumberSimulations*NumLoci);
  NumberSoFar := 0;

  for j:=1 to NSims do begin
    with frmRandCoal do begin
      memSimNum.Clear;
      memSimNum.Lines.Add(IntToStr(j));
      frmRandCoal.Repaint;
    end;
    MData := nil;
    SetLength(MData, Samples, NumLoci);
    Reset(MSatInput);
    for i := 0 to NumLoci-1 do begin
```

```
N := PopSize;
Read(MsatInput, locNumber);
Read(MsatInput, SizeLow);
Read(MsatInput, SizeHigh);
Read(MsatInput, repeatType);
Read(MsatInput, MSatMuRate);
Readln(MsatInput);  //skips to next line and next locus
RangeSize := SizeHigh-SizeLow;

repeat
  RandomAllele := Random(RangeSize);
  IntRandomAllele := Trunc(RandomAllele);
until (IntRandomAllele mod repeatType = 0);
  StartAllele1 := IntRandomAllele+SizeLow;

//Create identical starting msat alleles (those before mutation) in an array, with
//length/rows = # samples and width/columns = loci.

for q := 0 to (Samples)-1 do begin
  MData[q, i] := StartAllele1;
end;

CountCoalesce := 0;
CountCoalesce2 := 0;
CountMutation := 0;
CoalesceTime := 0;
CoalArrayLength := Samples-1;
SampleSize := Samples;
TreeMatrix := nil;
MutationMatrix := nil;
MutationScatter := nil;
MutLineages := nil;
MutLineages2 := nil;
NewickArray := nil;
SetLength(TreeMatrix, SampleSize-1, 3);
SetLength(MutationMatrix, 1000000, 2);
SetLength(MutationScatter, CoalArrayLength, CoalArrayLength);
SetLength(MutLineages, CoalArrayLength, Samples+1);
SetLength(MutLineages2, CoalArrayLength, Samples+1);
SetLength(NewickArray, Samples+1, Samples+1);

with frmRandCoal do begin
  ProgressBar1.FillColor := clBlue;
  ProgressBar1.Max := (Samples-1)*(NumLoci)*NumberSimulations;
end;

N := 2*N;

repeat //This repeat loop moves through one generation at a time, asking whether
there is a coalescent or mutation event.
```

```
        with frmRandCoal do begin
          If (chkFluctuatingN.checked) and (CoalesceTime > GStart) and (CoalesceTime
< GEnd) then begin
            StartN := N;
            Val(edtChangeN.Text, NFactor, code);
            AddInd := ((NFactor/100)*StartN);
            NextNReal := StartN+AddInd;
            NextN := Trunc(NextNReal);
            N := NextN;
          end;
        end;
        CoalesceTime := CoalesceTime+1;
        EndofRun := True;
        ChooseRandom(SampleSize,N);
        SortRandom(RandomkArray, SampleSize);
        MakeTree(SortedRandArray, SampleSize);
        MicrosatMutations(SampleSize, MsatMuRate);
        If CountCoalesce > CountCoalesce2 then begin
          SampleSize := Samples-CountCoalesce;
          NewickTree(SampleSize, Samples);
          with frmRandCoal do
            ProgressBar1.StepBy(1);
        end;
        CountCoalesce2 := CountCoalesce;
        if CountCoalesce < Samples-1 then
          EndofRun:=False;
      until EndofRun;

      TmrcaMSat[j-1,i] := CoalesceTime;

      //mutate the MData array with the relevant mutation model, before Mutation
Matrix
      //is reset on the simulation for the next locus
      //in the case of an infinite alleles model, write the number of alleles to a file

      IdentifyLineageClusters(Samples);
      MutateLineages(i);
      MutationDistribution(Samples);

      InfAllelesMSat[j-1,i]:=CountMutation+1;


      Append(NewickTreeFile);
      TrueNewickTree(Samples);
      StrValue := NewickArray2[Samples-1,0];
      Writeln(NewickTreeFile, StrValue + ';');
      CloseFile(NewickTreeFile);

      Append(TreeMatrixFile);
      Writeln(TreeMatrixFile, '#' + IntToStr(NumberSoFar+i+1));
```

```
    for tmf2 := 0 to Samples-2 do begin
     for tmf := 0 to 2 do begin
       value := TreeMatrix[tmf2, tmf];
       Write(TreeMatrixFile, IntToStr(value) + ' ');
     end;
     Writeln(TreeMatrixFile);
    end;
    CloseFile(TreeMatrixFile);

    Append(MutationMatrixFile);
    Writeln(MutationMatrixFile, '#' + IntToStr(NumberSoFar+i+1));
    for mmf := 0 to CountMutation-1 do begin
     for mmf2 := 0 to 1 do begin
       value := MutationMatrix[mmf,mmf2];
       Write(MutationMatrixFile, IntToStr(value) + ' ');
     end;
     Writeln(MutationMatrixFile);
    end;
    CloseFile(MutationMatrixFile);
    NumberMutationArray[NumberSoFar+i] := CountMutation;

   end; //end of number of loci loop

   NumberSoFar := NumberSoFar + (i);

   with frmRandCoal do begin
    if radAllele.checked or radStepwise.Checked and chkMSatOut.checked then
begin

     RandomChosenM := nil;
     SetLength(RandomChosenM, Samples);
     ts := 0;
     repeat
      RandomMSat := Random(Samples+1);
      RandomMSatInt := Trunc(RandomMSat);
      AlreadyUsed := False;
      tt := 0;
      repeat
       tempRandom := RandomChosenM[tt];
       if (tempRandom = RandomMSatInt) then
         AlreadyUsed := True;
       tt := tt+1;
      until (AlreadyUsed = True) or (tt = ts+1);
      if (AlreadyUsed = False) then begin
       RandomChosenM[ts] := RandomMSatInt;
       ts := ts+1;
      end;
     until (ts = Samples);

     Individ := 1;
```

```
      Append(MDataOut);
      for s:= 0 to Samples-1 do begin
        Allele1 := RandomChosenM[s];
        if (Odd(s) = False) and (s <> 0) then begin
          Individ := Individ+1;
        end;
        if (Odd(s) = False) or (s = 0) then begin
          StringName := IntToStr(Individ) + ' ';
          Write(MDataOut, StringName:10);
        end
        else begin
          Write(MDataOut, '          ');
        end;

        Write(MDataOut, ' ');
        for t:=1 to NumLoci do begin
          AlleleInt := MData[Allele1-1,t-1];
          Str(AlleleInt, AlleleStr);
          Write(MDataOut, AlleleStr);
          Write(MDataOut, ' ');
        end;
        Writeln(MDataOut);
      end;
      Writeln(MDataOut);
      CloseFile(MDataOut);

      //

    end;
  end;

  with frmRandCoal do begin
    If chkArlequin.Checked then begin
      Append(ArlequinBatch);
      Individ := 1;
      FileCounter := DirOut + '/Arlequin/sim_' + IntToStr(j) + '.arp';
      Writeln(ArlequinBatch, FileCounter);
      CloseFile(ArlequinBatch);
      AssignFile(ArlequinProject, FileCounter);
      Rewrite(ArlequinProject);
      Writeln(ArlequinProject, '[Profile]');
      Writeln(ArlequinProject, '   Title="Microsatellite data generated using coalescent
simulations"');
      Writeln(ArlequinProject, '   NbSamples=1');
      Writeln(ArlequinProject, '   GenotypicData=1');
      Writeln(ArlequinProject, '   GameticPhase=0');
      Writeln(ArlequinProject, '   DataType=MICROSAT');
      Writeln(ArlequinProject, '   LocusSeparator=WHITESPACE');
      Writeln(ArlequinProject, '[Data]');
      Writeln(ArlequinProject, ' [[Samples]]');
```

```
    Writeln(ArlequinProject, '   SampleName="Panmictic population"');
    ArlequinSampleSize := IntToStr(Samples div 2);
    Writeln(ArlequinProject, '   SampleSize=' + ArlequinSampleSize);
    Writeln(ArlequinProject, '   SampleData= {');

    for ai := 0 to Samples-1 do begin
      Allele1 := RandomChosenM[ai];
      if (Odd(ai) = False) and (ai <> 0) then begin
        Individ := Individ+1;
      end;
      if (Odd(ai) = False) or (ai = 0) then begin
        StringName := IntToStr(Individ) + ' ';
        Write(ArlequinProject, StringName:10);
        Write(ArlequinProject, ' 1 ');
      end
      else begin
        Write(ArlequinProject, '            ');
      end;
      for aj:=1 to NumLoci do begin
        AlleleInt := MData[Allele1-1,aj-1];
        Str(AlleleInt, AlleleStr);
        Write(ArlequinProject, AlleleStr);
        Write(ArlequinProject, ' ');
      end;
      Writeln(ArlequinProject);
    end;
    Writeln(ArlequinProject, '}');
    CloseFile(ArlequinProject);
  end;
end;

//be careful here. the following code changes the array MData, such that we
//can count alleles. I cannot copy the array to a new independent array since
//the copy function will only copy single dimension arrays. i could copy each
//column and susequently each allele in each locus to a new array, in a loop that
//counts the loci*2, but its much easier to simply change the MData array, and then
//not to use it again...ie. trash it.

with frmRandCoal do begin
  if radAllele.checked or radStepwise.Checked then begin
    SortedMData := nil;
    setLength(SortedMData, Samples, NumLoci);

    for ri := 1 to NumLoci do begin
      repeat
        sorted := True;
        for rj:=1 to (Samples)-1 do begin
          if (MData[rj-1, ri-1] > MData[rj, ri-1]) then begin
            tempInt := MData[rj-1, ri-1];
            MData[rj-1, ri-1] := MData[rj, ri-1];
```

```
          MData[rj,ri-1] := tempInt;
          sorted:=False;
        end;
      end;
    until sorted;

    for rl := 1 to Samples do begin
      tempInt := MData[rl-1,ri-1];
      SortedMData[rl-1,ri-1] := tempInt;
    end;

  end;

  NumberAlleles := nil;
  SetLength(NumberAlleles, NumLoci);
  AlleleFrequency := nil;
  SetLength(AlleleFrequency, Samples);
  for rm :=1 to NumLoci do begin
    rs := 0;
    CountAlleles := 1;
    AlleleCount := 1;
    for rn := 1 to (Samples-2) do begin
      tempInt := SortedMData[rn-1, rm-1];
      tempInt2 := SortedMData[rn, rm-1];
      if (tempInt = tempInt2) then begin
        AlleleCount := AlleleCount + 1;
      end;
      if (tempInt <> tempInt2) or (rn = Samples-1) then begin
        AlleleFrequency[rs] := AlleleCount/Samples;
        CountAlleles := CountAlleles+1;
        AlleleCount := 1;
        rs := rs+1;
      end;

    end;
    NumberAlleles[rm-1] := CountAlleles;
    FinAllelesMSat[j-1,rm-1] := CountAlleles;

    gd := 0;
    SumSquares := 0;
    AlleleFreq := AlleleFrequency[gd];
    while (AlleleFreq <> 0) do begin
      SumSquares := SumSquares + Power(AlleleFreq,2);
      gd := gd+1;
      AlleleFreq := AlleleFrequency[gd];
    end;
    GeneDiversity[j-1,rm-1] := (Samples/(Samples-1))*(1-SumSquares);
  end;
 end;
end;
```

```
  end; // end of number of simulations loop
  CloseFile(MSatInput);
  Outputs(NumberSimulations,NumLoci,Samples);
  TreeCounter := NumberSimulations*NumLoci;
  DrawTree;
end;

procedure DrawTree;

var
  originX : integer;
  originY : integer;
  radius : real;
  rad : integer;
  TimePlus, PointInterval, TimeCoalEvent, DrawTo : real;
  TimePlusAbs : real;
  TimePlusAbsInt : integer;
  DrawToInt : integer;
  i,j,l, m, o, p, q, r, s, a, b, c, d, e, f, g, h, k, t, u, w, x, y, z: integer;
  aa, ab, ac, ad, ae, af, ag, ah, ai, aj, ak, al, am, an, ao, ap, mi : integer;
  CoalCount : integer;
  Counter : integer;
  endTime, endTime2 : integer;
  value : integer;
  result, result2 : string;
  TimeGenerations : real;
  Time2N : real;
  PI, PlacePoint : integer;
  NumberSamples : integer;
  XSamples : array of integer;
  XSamples2 : array of integer;
  CoalescentTimes : array of integer;
  Descendent : integer;
  BranchID : integer;
  XMove, XMoveTwo : integer;
  TimeSingle : integer;
  TimeSingleA, TimeSingleB : real;
  NumberCoalGen : integer;
  FirstDescendent : integer;
  Time, NewTime : integer;
  TimeInt, TimePlusInt : integer;
  TimeLast : integer;
  TreeWidth : integer;
  TimeCoalEventInt : integer;
  BranchLength : integer;
  CoalTime : integer;
  CoalTimeInt : integer;
  CoalTimeCan : real;
  StartX, StartY : integer;
  EndX, EndY : integer;
```

```
  NextCoalescent : integer;
  NodeID : array of integer;
  NumberNodes : integer;
  NumberNodesInCoalescent : integer;
  NodePlaceX : integer;
  NumberGenCoals : integer;
  GenerationA, GenerationB : integer;
  next : integer;
  sum : integer;
  tester : integer;
  GenA : integer;
  TreeMatrix2 : array of integer;
  difference : integer;
  countLength : integer;
  NumberEvents : integer;
  TimeNowInt, Xmove2, YMove2, YMove : integer;
  Descendent1 : integer;
  TimeNow : real;
  PILength : integer;
  end3 : integer;
  TimePlus2 : integer;
  TimeLast2 : integer;
  TimePlusInt2 : integer;
  Temp : integer;
  DescendentPrev : integer;
  VerticalLimit : real;
  HorizontalLimit : real;
  StepSize : real;
  NumberCoalEvents : integer;
  MutLineages : array of array of integer;
  RowDescendent : integer;
  TestOne, TestTwo, TestThree, TestA, TestB, TestC, TestD : integer;
  MutationGen : integer;
  MutationGenAbs : integer;
  MutationGenReal : real;
  MutationLineage : integer;
  TimePrev : integer;


begin
  with frmRandCoal do begin
    Image1.Picture := nil;
    VerticalLimit := 490;
    HorizontalLimit := 500;

    //Draw dashed lines at each coalescent time, and assign the position to a matrix
called
    //CoalescentTimes.
    endTime := CoalArrayLength;
    CoalescentTimes := nil;
```

```
  SetLength(CoalescentTimes, endTime);
  StepSize := VerticalLimit/(N*3);
  for i:=1 to endTime do begin
   TimeSingle := TreeMatrix[i-1, 0];
   TimeLast := TreeMatrix[endTime-1, 0];
   TimeGenerations := TimeSingle;
   //Time2N := (TimeSingle/(2*N));
   TimeCoalEvent := VerticalLimit-(TimeSingle*StepSize);
   TimeCoalEventInt := Trunc(TimeCoalEvent);
   CoalescentTimes[i-1] := TimeCoalEventInt;
   with Image1,Canvas do begin
    value := Trunc(TimeCoalEvent);
    Pen.Style := psDash;
    Pen.Color := clBlack;
    Pen.Width := 1;
    Brush.Color := clWhite;
    MoveTo(10, value);
    //LineTo(1400, value);   //Draws dotted lines across tree at each coalescent event
    Font.Color := clBlue;
    Font.Size := 10;
    str(TimeGenerations:10:0, result);
    //str(Time2N:10:0, result2);
    TextOut(510, value-Font.Size, result)
   end;
  end;

 //Draw the tree from the top down, the top being the extant genes. First, we
determine
 //the number of coalescent events in each generation. Thereafter, the coalescent
events are drawn for each generation.
 //The matrix CoalGenerations assembles a matrix of each generation in which there
is one or more coalescent events
 //and the first individual (of each pair) involved in a coalescent event. Column 0
provides a very useful
 //value; i.e. the number of coalescent events at each generation - only important if
we change the code back to allowing
 //multiple coalescent events and polytomies - though drawing trees and keeping
track of mutations becomes difficult
 //when the assumption of one coalescent per generation is relaxed

  CoalGenerations := nil;
  SetLength(CoalGenerations, CoalArrayLength, CoalArrayLength);

  j := 1;
  r := 1;
  while j < CoalArrayLength+1 do begin
   q := 4;
   NumberGenCoals := 1;
   GenerationA := (TreeMatrix[j-1,0]);
   CoalGenerations[r-1, 2] := TreeMatrix[j-1, 1];
```

```
  CoalGenerations[r-1, 1] := GenerationA;
  for p:=j+1 to CoalArrayLength do begin
    GenerationB := TreeMatrix[p-1, 0];
    difference := (GenerationA-GenerationB);
    If difference = 0 then begin
      NumberGenCoals := NumberGenCoals+1;
      CoalGenerations[r-1, q-1] := TreeMatrix[p-1, 1];
      q := q+1;
    end;
  end;
  j:=j+NumberGenCoals;
  CoalGenerations[r-1, 0] := NumberGenCoals;
  r:=r+1;
end;

NumberCoalGen := r-1;
endTime2 := NumberCoalGen-1;
TimeNow := 490;
TimeNowInt := Trunc(TimeNow);
NumberSamples := CoalArrayLength+1;
PointInterval := (HorizontalLimit/(NumberSamples));
PI := Trunc(PointInterval);
TreeWidth := (NumberSamples-1)*PI;
PlacePoint := 20+PI;

//Get your starting points and write them to a matrix called XSamples
XSamples := nil;
SetLength(XSamples, NumberSamples);
for c:=1 to NumberSamples do begin
  with Image1,Canvas do begin
    Pen.Style := psSolid;
    Radius := 3.0;
    rad := Trunc(Radius);
    Brush.Style := bsSolid;
    Brush.Color := clBlue;
    XSamples[c-1] := PlacePoint;
    PlacePoint := PlacePoint+PI;
  end;
end;

for a:=1 to NumberCoalGen do begin
  If (a = 1) then begin
    TimePrev := 0;
  end
  else begin
    TimePrev := CoalGenerations[a-2, 1];
  end;

  TimeLast := CoalGenerations[endTime2, 1];
  TimePlus := CoalGenerations[a-1, 1];
```

```
TimePlusAbsInt := Trunc(TimePlus);
TimePlus := VerticalLimit-(TimePlus*StepSize);
TimePlusInt := Trunc(TimePlus);
NumberEvents := CoalGenerations[a-1, 0];
XSamples2 := nil;
SetLength(XSamples2, NumberSamples);

for t:=1 to NumberSamples do begin
  XSamples2[t-1] := XSamples[t-1];
end;

for m:=1 to NumberSamples do begin
  PlacePoint := XSamples[m-1];
  with Image1,Canvas do begin
    Pen.Style := psSolid;
    OriginX := PlacePoint;
    OriginY := TimeNowInt;
    MoveTo(OriginX, OriginY);
    LineTo(OriginX, TimePlusInt);
  end;
end;

  //this tree drawing code can handle up to four samples coalescing into 1 in a single
event.
  //to be able to handle more you need to keep checking the subsequent descendent
against the first!
  //you can probably do this with a controlled forward loop...but its not neccessary
since I have now
  //forced only one coalescent event per generation!

  s:=2;
  u := 0;
  x := 0;

  while u < NumberSamples do begin
    Descendent1 := CoalGenerations[a-1, s];
    if Descendent1 = (u) then begin
      with Image1,Canvas do begin
        XMove := XSamples[Descendent1];
        MoveTo(XMove, TimePlusInt);
        XMoveTwo := XMove + ((XSamples[Descendent1+1]-XMove) div 2);
        DrawToInt := XSamples[Descendent1+1];
        LineTo(DrawToInt, TimePlusInt);
        XSamples2[Descendent1-(s-2)] := XMoveTwo;
        s:= s+1;
        u := u+2;
        DescendentPrev := Descendent1;
        Descendent1 := CoalGenerations[a-1, s];
        if Descendent1 = DescendentPrev+1 then begin
          XMove := XSamples[Descendent1+1];
```

```
            MoveTo(XMove, TimePlusInt);
            XMoveTwo := XMove - ((XMove - XSamples[DescendentPrev]) div 2);
            DrawToInt := XSamples[DescendentPrev];
            LineTo(DrawToInt, TimePlusInt);
            XSamples2[DescendentPrev+1-(s-2)] := XMoveTwo;
            s := s+1;
            u := u+1;
            DescendentPrev := Descendent1;
            Descendent1 := CoalGenerations[a-1, s];
            if Descendent1 = DescendentPrev+1 then begin
              XMove := XSamples[Descendent1+1];
              MoveTo(XMove, TimePlusInt);
              XMoveTwo := XMove - ((XMove - XSamples[DescendentPrev]) div 2);
              DrawToInt := XSamples[DescendentPrev];
              LineTo(DrawToInt, TimePlusInt);
              XSamples2[DescendentPrev+1-(s-2)] := XMoveTwo;
              s := s+1;
              u := u+1;
            end;
          end;
        end;
      end
      else begin
        XSamples2[(u)-(s-2)] := XSamples[u];
        u := u+1;
      end;
    end;

    if chkDrawMut.checked then begin
      if CountMutation > 0 then begin
        for mi := 0 to CountMutation-1 do begin
          MutationGenAbs := MutationMatrix[mi, 0];
          if (MutationGenAbs <= TimePlusAbsInt) and (MutationGenAbs > TimePrev)
then begin //i.e the mutation has occured b4 next coalescent event
            MutationLineage := MutationMatrix[mi, 1];
            XMove := XSamples[MutationLineage];
            MutationGenReal := VerticalLimit-(MutationGenAbs*StepSize);
            MutationGen := Trunc(MutationGenReal);
            with Image1,Canvas do begin
              MoveTo(XMove-4, MutationGen);
              Pen.Style := psSolid;
              Pen.Color := clRed;
              Pen.Width := 3;
              LineTo(XMove+4, MutationGen);
              MoveTo(5, MutationGen);
              Font.Color := clRed;
              Font.Size := 10;
              str(MutationGenAbs, result);
              TextOut(5, MutationGen-Font.Size, result);
              Pen.Width := 1;
```

```
            Pen.Color := clBlack;
           end;
          end;
         end;
        end;
       end;

       for w :=1 to NumberSamples do begin
        XSamples[w-1] := XSamples2[w-1];
       end;

       TimeNowInt := TimePlusInt;
       NumberSamples := NumberSamples-(NumberEvents);
     end;
     with Image1,Canvas do begin
       Font.Color := clBlack;
       Font.Size := 12;
       TextOut(250, 510, 'TMRCA = '+IntToStr(TimeLast));
     end;
   end;
end;

procedure LogFile(kL,NL,StartNL,SeqLengthL,GrowthStartL,GrowthEndL,
NumberSimulationsL: integer; StartTimeL, EndTimeL : TDateTime;
SeqRateL,FreqAL,FreqCL,FreqGL,FreqTL : real);

var
  logfile : TextFile;
  MSatIn : TextFile;
  strVal : string;
  model : string;
  titvReal : real;
  titv : string[4];
  invarstr : string;
  invarReal : real;
  shapestr : string;
  shapeReal : real;
  strA, strC, strG, strT : string;
  strTReal : real;
  strAReal : real;
  strCReal : real;
  strGReal : real;
  code : integer;
  valueReal : real;
  Result : string;
  i : integer;
  NumLociL : integer;
  locNumberInt, SizeLowInt, SizeHighInt, repeatTypeInt : integer;
  MSatMuRateReal : real;
  MSatMuRateStr : string;
```

```
  vkL : real;
  VkLStr : string;
  NChangeReal : real;
  NChangeStr : string;



begin
  with frmRandCoal do begin
    AssignFile(logfile, DirOut + '/' + edtLogFile.Text);
    Rewrite(logfile);
    Writeln(logfile, 'Coalescent Simulations generated with RandoCoal0.2b: LogFile');
    Writeln(logfile,
'************************************************************');
    Writeln(logfile);
    Writeln(logfile, DateToStr(Date));

    Writeln(logfile, 'Started at: ' + TimeToStr(StartTimeL));
    Writeln(logfile, 'Finished at: ' + TimeToStr(EndTimeL));
    Writeln(logfile);
    Writeln(logfile, 'Coalescent Parameters');
    Writeln(logfile, '**********');
    Writeln(logfile);
    if (chkFluctuatingN.Checked) then begin
      Writeln(logfile, 'Starting population Size: ' + IntToStr(StartNL));
      Writeln(logfile, 'Final population Size: ' + IntToStr(NL));
    end
    else begin
      Writeln(logfile, 'Population size: ' + IntToStr(NL));
    end;

    Writeln(logfile, 'Number of samples: ' + IntToStr(kL));
    Writeln(logfile, 'Number of simulations: ' + IntToStr(NumberSimulationsL));


    Writeln(logfile);
    Writeln(logfile, 'Mutation Parameters');
    Writeln(logfile, '**********');
    Writeln(logfile);

    if (chkMsats.Checked) then begin
      Writeln(logfile, 'Datatype: Microsatellites');
      AssignFile(MsatIn, DirOut + '/' + edtMsatIn.Text);
      Reset(MsatIn);
      Val(edtNumLoci.Text, NumLociL, code);
      Writeln(logfile, 'Number of loci: ' + IntToStr(NumLociL));
      if (radInfiniteAlleles.Checked) then
        Writeln(logfile, 'Mutation model: infinite allele')
      else if (radStepwise.Checked) then
        Writeln(logfile, 'Mutation model: Stepwise')
```

```
    else if (radAllele.Checked) then
      Writeln(logfile, 'Mutation model: Random allele');
    Writeln(logfile);
    Writeln(logfile, 'Locus #   Size Range   Repeat Type      Mutation Rate');
    Writeln(logfile, '*******   **********   ***********      *************');

    for i := 0 to NumLociL-1 do begin
      Read(MsatIn, locNumberInt);
      Read(MsatIn, SizeLowInt);
      Read(MsatIn, SizeHighInt);
      Read(MsatIn, repeatTypeInt);
      Read(MsatIn, MSatMuRateReal);
      Readln(MsatIn);  //skips to next line and next locus
      Str(MsatMuRateReal, MSatMuRateStr);
      Write(logfile, IntToStr(locNumberInt):3, IntToStr(SizeLowInt):10,
IntToStr(SizeHighInt):5, IntToStr(repeatTypeInt):8);
      Write(logfile, '          ' + MSatMuRateStr);
      Writeln(logfile);
    end;
    CloseFile(MSatIn);

    Writeln(logfile, 'Output Files');
    Writeln(logfile, '*************');
    Writeln(logfile);
    Writeln(logfile, 'Tmrca and diversity indices: ' + edtTmrca.Text);
    Writeln(logfile, 'Newick tree output file: ' + edtNewickTrees.Text);
    if chkMSatOut.Checked then
      Writeln(logfile, 'Microsatellite profiles generated: ' + edtMSatData.Text);
    Writeln(logfile, 'Distribution of coalescence times with genealogy length: ' +
edtCoalDistrib.Text);
    Writeln(logfile, 'Distribution of mutation times with genealogy length: ' +
edtMutDistrib.Text);

  end
  else begin
    Writeln(logfile, 'Datatype: DNA sequence');
    Str(SeqRateL, strVal);

    Writeln(logfile, 'Mutation rate: ' + strVal);
    Writeln(logfile, 'Sequence Length: ' + IntToStr(SeqLengthL));

    if (chkSeqInput.checked) then begin
      Writeln(logfile, 'Root sequence provided in ' + edtSeqFile.Text);
    end
    else begin
      Writeln(logfile, 'Root sequence generated at random');
    end;

    If (radJC69.Checked) then begin
      model := 'JC69';
```

```
  Writeln(logfile, 'Substitution model: ' + model);
  If (chkInvarSites.checked) then begin
    Val(edtI.Text, invarReal, code);
    Str(invarReal:0:3, invarstr);
    Writeln(logfile, 'Proportion of invariable sites: ' + invarstr);
  end;
  If (chkShapeParameter.checked) then begin
    Val(edtAlpha.Text, shapeReal, code);
    Str(shapeReal:0:3, shapestr);
    Writeln(logfile, 'Among-site rate variation shape parameter: ' + shapestr);
  end;
end;

if (radF81.Checked) then begin
  model := 'F81';
  Writeln(logfile, 'Substitution model: ' + model);
  Write(logfile, 'A:C:G:T = ');
  Str(FreqAL:0:2, strA);
  Str(FreqCL:0:2, strC);
  Str(FreqGL:0:2, strG);
  Str(FreqTL:0:2, strT);
  Write(logfile, strA + ':' + strC + ':' + strG + ':' + strT);

  Writeln(logfile);
  If (chkInvarSites.checked) then begin
    Val(edtI.Text, invarReal, code);
    Str(invarReal:0:3, invarstr);
    Writeln(logfile, 'Proportion of invariable sites: ' + invarstr);
  end;
  If (chkShapeParameter.checked) then begin
    Val(edtAlpha.Text, shapeReal, code);
    Str(shapeReal:0:3, shapestr);
    Writeln(logfile, 'Among-site rate variation shape parameter: ' + shapestr);
  end;
end;

if (radK2P.Checked) then begin
  model := 'Kimura 2-parameter';
  Writeln(logfile, 'Substitution model: ' + model);
  Val(edtTiTv.Text, titvReal, code);
  Str(titvReal:0:2, titv);
  Writeln(logfile, 'TiTv ratio: ' + titv);
  If (chkInvarSites.checked) then begin
    Val(edtI.Text, invarReal, code);
    Str(invarReal:0:3, invarstr);
    Writeln(logfile, 'Proportion of invariable sites: ' + invarstr);
  end;
  If (chkShapeParameter.checked) then begin
    Val(edtAlpha.Text, shapeReal, code);
    Str(shapeReal:0:3, shapestr);
```

```
        Writeln(logfile, 'Among-site rate variation shape parameter: ' + shapestr);
      end;
    end;

    if (radHKY85.Checked) then begin
      model := 'HKY85';
      Writeln(logfile, 'Substitution model: ' + model);
      Val(edtTiTv.Text, titvReal, code);
      Str(titvReal:0:2, titv);
      Writeln(logfile, 'TiTv ratio: ' + titv);
      Write(logfile, 'A:C:G:T = ');
      Str(FreqAL:0:2, strA);
      Str(FreqCL:0:2, strC);
      Str(FreqGL:0:2, strG);
      Str(FreqTL:0:2, strT);
      Write(logfile, strA + ':' + strC + ':' + strG + ':' + strT);
      Writeln(logfile);
      If (chkInvarSites.checked) then begin
        Val(edtI.Text, invarReal, code);
        Str(invarReal:0:3, invarstr);
        Writeln(logfile, 'Proportion of invariable sites: ' + invarstr);
      end;
      If (chkShapeParameter.checked) then begin
        Val(edtAlpha.Text, shapeReal, code);
        Str(shapeReal:0:3, shapestr);
        Writeln(logfile, 'Among-site rate variation shape parameter: ' + shapestr);
      end;
    end;
    Writeln(logfile);
    Writeln(logfile, 'Output Files');
    Writeln(logfile, '************');
    Writeln(logfile);
    Writeln(logfile, 'Tmrca and diversity indices: ' + edtTmrca.Text);
    Writeln(logfile, 'Newick tree output file: ' + edtNewickTrees.Text);
    if chkSeqOut.Checked then
      Writeln(logfile, 'Sequence data output file: ' + edtSeqOut.Text);
    Writeln(logfile, 'Distribution of coalescence times with genealogy length: ' +
edtCoalDistrib.Text);
    Writeln(logfile, 'Distribution of mutation times with genealogy length: ' +
edtMutDistrib.Text);


  end;

  if (chkVarRepSuc.Checked) or (chkFluctuatingN.Checked) then begin
    Writeln(logfile);
    Writeln(logfile, 'Demographics');
    Writeln(logfile, '************');
    Writeln(logfile);
    if (chkVarRepSuc.Checked) then begin
```

```
        Val(edtVarRepSuc.Text, VkL, code);
        Str(Vkl:0:3, VklStr);
        Writeln(logfile, 'Variance in reproductive success: ' + VkLStr);
      end;

      if (chkFluctuatingN.Checked) then begin
        Writeln(logfile, 'Population growth starts at generation: ' +
IntToStr(GrowthStartL));
        Writeln(logfile, 'Population growth ends at generation: ' +
IntToStr(GrowthEndL));
        Val(edtChangeN.Text, NChangeReal, code);
        Str(NChangeReal:0:2, NChangeStr);
        Writeln(logfile, 'Population size changes by ' + NChangeStr + '% each
generation');
      end;
    end;
    CloseFile(logfile);
  end;
end;

procedure Outputs(NumberSims, NLoci, NumSamples: integer);

var
  outfileAll : textfile;
  outfileAllM : textfile;
  Intvalue : integer;
  realValue : real;
  realString : string;
  i,j : integer;
  outfileSeq : textFile;
  StringReal : string;


begin
  with frmRandCoal do begin
    if chkMsats.Checked then begin
      AssignFile(outfileAllM, DirOut + '\' + edtTmrca.Text);
      Rewrite(outfileAllM);
      Write(outfileAllM, 'Simulation #, ');
      for j := 1 to NLoci do begin
        Write(outfileAllM, 'Tmrca:Loc ' + IntToStr(j) + ', ');
        Write(outfileAllM, 'Inf a:Loc ' + IntToStr(j) + ', ');
        Write(outfileAllM, 'Fin a:Loc ' + IntToStr(j) + ', ');
        if chkMSatDiversity.Checked then begin
          Write(outfileAllM, 'Allelic Diversity:Loc ' + IntToStr(j) + ', ');
          Write(outfileAllM, 'Gene Diversity:Loc ' + IntToStr(j) + ', ');
        end;
      end;
      Writeln(outfileAllM);
```

```
    for i := 1 to NumberSims do begin
      Write(outfileAllM, IntToStr(i) + ', ');
      for j := 1 to NLoci do begin
        Intvalue := TmrcaMSat[i-1,j-1];
        Write(outfileAllM, IntToStr(Intvalue) + ', ');
        Intvalue := InfAllelesMSat[i-1,j-1];
        Write(outfileAllM, IntToStr(Intvalue) + ', ');
        Intvalue := FinAllelesMSat[i-1,j-1];
        Write(outfileAllM, IntToStr(Intvalue) + ', ');
        if chkMsatDiversity.Checked then begin
          Intvalue := FinAllelesMsat[i-1,j-1];
          Realvalue := Intvalue/(NumSamples);
          Str(RealValue:6:3, StringReal);
          Write(outfileAllM, StringReal + ', ');
          Realvalue := GeneDiversity[i-1,j-1];
          Str(Realvalue:6:3, StringReal);
          Write(outfileAllM, StringReal + ', ');
        end;
      end;
      Writeln(outfileAllM);
    end;
    CloseFile(outfileAllM);
  end
  else begin
    AssignFile(outfileAll, DirOut + '\' + edtTmrca.Text);
    Rewrite(outfileAll);
    if chkSeqDiversity.Checked then
      Writeln(outfileAll, 'Simulation#, Tmrca, Infinite s, Finite s, Allelic Div,
Nucleotide Div')
    else
      Writeln(outfileAll, 'Simulation #, Tmrca, Infinite s, Finite s');
    for i := 1 to NumberSims do begin
      Write(outfileAll, IntToStr(i) + ', ');
      Intvalue := Tmrca[i-1];
      Write(outfileAll, IntToStr(Intvalue) + ', ');
      Intvalue := InfAlleles[i-1];
      Write(outfileAll, IntToStr(Intvalue) + ', ');
      Intvalue := FinAlleles[i-1];
      Write(outfileAll, IntToStr(Intvalue));
      if chkSeqDiversity.Checked then begin
        Realvalue := AllelicDiversity[i-1];
        str(RealValue:4:5, StringReal);
        Write(outfileAll, ', ' + StringReal + ', ');
        Realvalue := NucleotideDiversity[i-1];
        str(RealValue:4:5, StringReal);
        Write(outfileAll, StringReal);
      end;
      Writeln(outfileAll);
    end;
    CloseFile(outfileAll);
```

```
    end;
  end;
end;


procedure TfrmRandCoal.RunSimulation(Sender: TObject);

var
  k, code, j, l : integer;
  tmf, tmf2, mmf, mmf2 : integer;
  ai, aj : integer;
  m, p : integer;
  o, s : integer;
  value : integer;
  StrValue : string;
  result : string;
  //result1, result2 : string;
  //value1, value2 : integer;
  i,h,q,r,v,w,x,y : integer;
  rs : integer;
  //LastTime : integer;
  //Sequence : array of char;
  SeqLength : integer;
  RandomChar : integer;
  base : char;
  infile : TextFile;
  outfile, outfileSeq : TextFile;
  outfileSeqLength : TextFile;
  t, u : integer;
  seed : integer;
  resultString : string;
  ab, ac : integer;
  GrowthStart : integer;
  GrowthEnd : integer;
  Vk : real;
  NReal : real;
  PropInVar : real;
  SeqLengthReal : real;
  FreqA, FreqC, FreqG, FreqT : real;
  alpha : real;
  Nst : integer;
  StringShape : string;
  StartTime, EndTime : TDateTime;
  NBeforeGrowth : integer;
  RandomSequence : real;
  RandomSequenceInt : integer;
  RandomChosen : array of integer;
  AlreadyUsed : boolean;
  tempRandom : integer;
  rt,rc : integer;
```

```
  Individ : integer;
  RandomInd : integer;
  StringName : string;
  mk : integer;

begin
  StartTime := Time;
  NumberSimulations := 1;
  Randomize;
  Val(edtSampleSize.Text, k, code);
  Val(edtPopulationSize.Text, N, code);
  NBeforeGrowth := N;
  GrowthStart := 0;
  GrowthEnd := 0;

  with ProgressBar1 do begin
    Position := 0;
  end;

  AssignFile(outfileCoalDis, DirOut + '\' + edtCoalDistrib.Text);
  Rewrite(outfileCoalDis);

  AssignFile(outfileMuDis, DirOut + '\' + edtMutDistrib.Text);
  Rewrite(outfileMuDis);

  If (k > 0) and (N > 0) then begin
    If chkFluctuatingN.Checked then begin
      Val(edtGrowthStart.Text, GrowthStart, code);
      Val(edtGrowthEnd.Text, GrowthEnd, code);
    end;

    If chkMultipleSims.Checked then begin
      Val(edtNumberSimulations.Text, NumberSimulations, code);
    end;

    If chkVarRepSuc.checked then begin
      Val(edtVarRepSuc.Text, Vk, code);
      Val(edtMeanOffsp.Text, Mk, code);
      if chkDiploid.checked then begin
        NReal := ((N*Mk)-1)/((Mk-1+(Vk/Mk)));
        N := Trunc(NReal);
      end
      else begin
        NReal := (((N*Mk)-1))/((Mk-1+(Vk/Mk)))/2;
        N := Trunc(NReal);
      end;
    end;

    If chkScatterMut.Checked and chkSeqOut.checked then begin
      AssignFile(outfileSeq, DirOut + '\' + edtSeqOut.Text);
```

```
    Rewrite(outfileSeq);
  end;

  If chkArlequin.Checked and (chkSeqOut.Checked or chkMSatOut.Checked) then
begin
    CreateDir(DirOut + '\' + 'Arlequin');
    AssignFile(ArlequinBatch, DirOut + '\' + 'ArlequinBatch.arp');
    Rewrite(ArlequinBatch);
  end;

  If chkMsats.checked then begin
    MicrosatModel(k,N,GrowthStart,GrowthEnd,NumberSimulations);
  end
  else begin

    Val(edtMutationRate.Text, MuRate, code);
    Val(edtSeqLength.Text, SeqLength, code);

    If chkInvarSites.checked then begin
      Val(edtI.Text, PropInVar, code);
      SeqLengthReal := SeqLength-(Seqlength*(PropInVar));
      SeqLength := Trunc(SeqLengthReal);
    end;

    If chkShapeParameter.checked then begin
      Val(edtAlpha.Text, alpha, code);
    end;

    If chkDiploid.checked then begin
      N := 2*N;
    end;

    GeneMuRate(MuRate, SeqLength);
    kSamples := k;

    If chkSeqInput.Checked then begin
      AssignFile(infile, DirOut + '\' + edtSeqFile.Text);
      Reset(infile);
      m := 0;
      while not(Eof(infile)) do begin
        Read(infile, base);
        m := m+1;
      end;
      Sequence := nil;
      SeqLength := m;
      SetLength(Sequence, SeqLength);
      Reset(infile);
      for p := 0 to m-1 do begin
        Read(infile, base);
        Sequence[p] := base;
```

```
      end;
      CloseFile(infile);
      SeqLength := m-1;
    end;

  //currently if we run multiple sims, each sim starts with the same randomly
generated sequence or input sequence provided.
  //we may want to change the random starting sequence each sim, but i'm not sure
why this would be more appropriate than
  //the current model?

    If (chkSeqInput.Checked = False) then begin
      Sequence := nil;
      SetLength(Sequence, SeqLength);

      If (radF81.Checked) or (radHKY85.Checked) then begin
        Val(edtFreqA.Text, FreqA, code);
        Val(edtFreqC.Text, FreqC, code);
        Val(edtFreqG.Text, FreqG, code);
        FreqT := 1-(FreqA+FreqC+FreqG);
        for h := 0 to SeqLength-1 do begin
          RandomChar := Random(100);
          if (RandomChar >= 0) and (RandomChar < FreqA*100) then
            Sequence[h] := 'a';
          if (RandomChar >= FreqA*100) and (RandomChar < (FreqA+FreqC)*100)
then
            Sequence[h] := 'c';
          if (RandomChar >= (FreqA+FreqC)*100) and (RandomChar <
(FreqA+FreqC+FreqG)*100) then
            Sequence[h] := 'g';
          if (RandomChar >= (FreqA+FreqC+FreqG)*100) and (RandomChar <= 100)
then
            Sequence[h] := 't';
        end;
      end
      else begin
      for h := 0 to SeqLength-1 do begin
        RandomChar := Random(4);
        if (RandomChar >= 0) and (RandomChar < 1) then
          Sequence[h] := 'a';
        if (RandomChar >= 1) and (RandomChar < 2) then
          Sequence[h] := 'c';
        if (RandomChar >= 2) and (RandomChar < 3) then
          Sequence[h] := 'g';
        if (RandomChar >= 3) and (RandomChar <= 4) then
          Sequence[h] := 't';
        end;
      end;
    end;
```

```
AssignFile(NewickTreeFile, DirOut + '\' + edtNewickTrees.Text);
Rewrite(NewickTreeFile);

AssignFile(TreeMatrixFile, DirOut + '\' + 'TreeMatFile.txt');
Rewrite(TreeMatrixFile);
AssignFile(MutationMatrixFile, DirOut + '\' + 'MutMatFile.txt');
Rewrite(MutationMatrixFile);

Tmrca := nil;
SetLength(Tmrca, NumberSimulations);
InfAlleles := nil;
SetLength(InfAlleles, NumberSimulations);
FinAlleles := nil;
SetLength(FinAlleles, NumberSimulations);
GammaPerSiteAr := nil;
SetLength(GammaPerSiteAr, NumberSimulations);
GammaGeneMuAr := nil;
SetLength(GammaGeneMuAr, NumberSimulations);
AllelicDiversity := nil;
SetLength(AllelicDiversity, NumberSimulations);
NucleotideDiversity := nil;
SetLength(NucleotideDiversity, NumberSimulations);
NumberMutationArray := nil;
SetLength(NumberMutationArray, NumberSimulations);

for i :=1 to NumberSimulations do begin
  frmRandCoal.Repaint;
  memSimNum.Clear;
  memSimNum.Lines.Add(IntToStr(i));
  CountCoalesce := 0;
  CountCoalesce2 := 0;
  CountMutation := 0;
  CoalesceTime := 0;
  CoalArrayLength := k-1;
  SampleSize := k;
  TreeMatrix := nil;
  MutationMatrix := nil;
  MutationScatter := nil;
  MutLineages := nil;
  MutLineages2 := nil;
  NewickArray := nil;
  NewickArray2 := nil;
  SetLength(TreeMatrix, SampleSize-1, 3);
  SetLength(MutationMatrix, 1000000, 2);
  SetLength(MutationScatter, CoalArrayLength, CoalArrayLength);
  SetLength(MutLineages, CoalArrayLength, k+1);
  SetLength(MutLineages2, CoalArrayLength, k+1);
  SetLength(NewickArray, k+1, k+1);
  SetLength(NewickArray2, k+1, k+1);
  ProgressBar1.FillColor := clblue;
```

```
      ProgressBar1.Max := (k-1)*NumberSimulations;


    //Create identical starting sequences (those before mutation) in an array, with
    //length/rows = # characters and width/columns = # samples. Therefore, character
5
    //in sample 3 will be mutated as Data [4, 2] := ...

      Data := nil;
      SetLength(Data, SeqLength, k);
      for q := 0 to k-1 do begin
        for r := 0 to SeqLength-1 do begin
          Data[r, q] := Sequence[r];
        end;
      end;



      repeat //This repeat loop moves through one generation at a time, asking whether
there is a coalescent or mutation event.
        If (chkFluctuatingN.checked) and (CoalesceTime > GrowthStart) and
(CoalesceTime < GrowthEnd) then begin
          StartN := N;
          Val(edtChangeN.Text, NFactor, code);
          AddInd := ((NFactor/100)*StartN);
          NextNReal := StartN+AddInd;
          NextN := Trunc(NextNReal);
          N := NextN;
        end;

        CoalesceTime := CoalesceTime+1;
        EndofRun := True;
        ChooseRandom(SampleSize,N);
        SortRandom(RandomkArray, SampleSize);
        MakeTree(SortedRandArray, SampleSize);  //CountCoalesce is incremented if
there is a coalescent event
        If chkShapeParameter.checked then
          GammaMutations(SampleSize,SeqLength,i,alpha,MuRate)
        else
        Mutations(SampleSize);
        If CountCoalesce > CountCoalesce2 then begin
          SampleSize := k-CountCoalesce;
          NewickTree(SampleSize, k);
          ProgressBar1.StepBy(1);
        end;
        CountCoalesce2 := CountCoalesce;
        if CountCoalesce < k-1 then
          EndofRun:=False;
      until EndofRun;
```

```
pseudoNewick := NewickArray[k-1, 0];
pseudoNewickArray := nil;
SetLength(pseudoNewickArray, Length(pseudoNewick));


If chkScatterMut.checked then begin
  IdentifyLineageClusters(k);
  MutateLineages(SeqLength);
end;

MutationDistribution(k);
TotalSegSitesFin := 0;
DiversityIndices(SeqLength, k, i);

Append(NewickTreeFile);
TrueNewickTree(k);
StrValue := NewickArray2[k-1,0];
Writeln(NewickTreeFile, StrValue + ';');
CloseFile(NewickTreeFile);

If chkScatterMut.checked and chkSeqOut.Checked then begin


  //If the gene is a diploid gene we must randomly scramble the sequences
  //to assemble genotypes. Else if we leave it ordered genotypes will
  //be arranged as in the genealogy, and alleles within a genotype will
  //more closely related than alleles between genotypes. The same applies
  //to microsat data


  If chkDiploid.Checked then begin
    RandomChosen := nil;
    SetLength(RandomChosen, k);
    rs := 0;
    repeat
      RandomSequence := Random(k+1);
      RandomSequenceInt := Trunc(RandomSequence);
      AlreadyUsed := False;
      rt := 0;
      repeat
        tempRandom := RandomChosen[rt];
        if (tempRandom = RandomSequenceInt) then
          AlreadyUsed := True;
        rt := rt+1;
      until (AlreadyUsed = True) or (rt = rs+1);
      if (AlreadyUsed = False) then begin
        RandomChosen[rs] := RandomSequenceInt;
        rs := rs+1;
      end;
    until (rs = k);
```

```
        If rdgOutput.Items[rdgOutput.ItemIndex] = 'Fasta' then begin
          Append(outfileSeq);
          Writeln(outfileSeq, IntToStr(k) + ' ' + IntToStr(SeqLength));
          Individ := 1;
          for v := 0 to k-1 do begin
            RandomInd := RandomChosen[v];
            if (Odd(v) = False) and (v <> 0) then
              Individ := Individ+1;
            StringName := IntToStr(Individ) + '_' + IntToStr(i) + ' ';
            Write(outfileSeq, StringName:10);
            for w := 0 to SeqLength-1 do begin
              base := Data[w,RandomInd-1];
              Write(outfileSeq, UpperCase(base));
            end;
            Writeln(outfileSeq);
          end;
          WriteLn(outfileSeq);
          CloseFile(outfileSeq);
        end;

        If rdgOutput.Items[rdgOutput.ItemIndex] = 'Clustal' then begin
          Append(outfileSeq);
          Individ := 1;
          for v := 0 to k-1 do begin
            RandomInd := RandomChosen[v];
            if (Odd(v) = False) and (v <> 0) then
              Individ := Individ+1;
            Writeln(outfileSeq, '>' + 'Ind_' + IntToStr(Individ)+ '_' +
IntToStr(RandomInd)+'_'+IntToStr(i));
            for w := 0 to SeqLength-1 do begin
              base := Data[w,RandomInd-1];
              Write(outfileSeq, UpperCase(base));
            end;
            Writeln(outfileSeq);
          end;
          WriteLn(outfileSeq);
          CloseFile(outfileSeq);
        end;

        If rdgOutput.Items[rdgOutput.ItemIndex] = 'MrBayes' then begin
          Append(outfileSeq);
          Individ := 1;
          Writeln(outfileSeq,'#NEXUS');
          Writeln(outfileSeq);
          Writeln(outfileSeq,'BEGIN DATA;');
          Writeln(outfileSeq, 'DIMENSIONS ' + 'NTAX=' + IntToStr(k)+ ' NCHAR='
+ IntToStr(SeqLength) + ';');
          Writeln(outfileSeq, 'FORMAT DATATYPE=DNA;');
          Writeln(outfileSeq, 'MATRIX');
```

```
            for v := 0 to k-1 do begin
              RandomInd := RandomChosen[v];
              if (Odd(v) = False) and (v <> 0) then
                Individ := Individ+1;
              Writeln(outfileSeq, 'Ind' + IntToStr(Individ) + '_' +
IntToStr(RandomInd)+'_'+IntToStr(i));
              for w := 0 to SeqLength-1 do begin
                base := Data[w,RandomInd-1];
                Write(outfileSeq, UpperCase(base));
              end;
              Writeln(outfileSeq);
            end;
            Writeln(outfileSeq, ';');
            Writeln(outfileSeq, 'end;');
            WriteLn(outfileSeq);
            Writeln(outfileSeq, 'begin mrbayes;');
            Writeln(outfileSeq, 'set autoclose=yes;');
            if radJC69.Checked then
              Nst := 1
            else Nst := 2;
            if chkShapeParameter.Checked then
              StringShape := 'rates=gamma'
            else StringShape := '';
            Writeln(outfileSeq, 'lset nst=' + IntToStr(Nst) + ' ' + StringShape + ';');
            Writeln(outfileSeq, 'mcmc ngen=1000 printfreq=100 samplefreq=10
nchains=4 savebrlens=yes;');
            Writeln(outfileSeq, 'end;');
            CloseFile(outfileSeq);
          end;

          If rdgOutput.Items[rdgOutput.ItemIndex] = 'Nexus' then begin
            Append(outfileSeq);
            Individ := 1;
            Writeln(outfileSeq,'#NEXUS');
            Writeln(outfileSeq);
            Writeln(outfileSeq,'BEGIN DATA;');
            Writeln(outfileSeq, 'DIMENSIONS ' + 'NTAX=' + IntToStr(k)+ ' NCHAR='
+ IntToStr(SeqLength) + ';');
            Writeln(outfileSeq, 'FORMAT DATATYPE=DNA;');
            Writeln(outfileSeq, 'MATRIX');
            for v := 0 to k-1 do begin
              RandomInd := RandomChosen[v];
              if (Odd(v) = False) and (v <> 0) then
                Individ := Individ+1;
              Writeln(outfileSeq, 'Ind_' + IntToStr(Individ) + '_' +
IntToStr(RandomInd)+'_'+IntToStr(i));
              for w := 0 to SeqLength-1 do begin
                base := Data[w,RandomInd-1];
                Write(outfileSeq, UpperCase(base));
              end;
```

```
        Writeln(outfileSeq);
      end;
      Writeln(outfileSeq, ';');
      Writeln(outfileSeq, 'end;');
      WriteLn(outfileSeq);
      CloseFile(outfileSeq);
    end;

    If chkArlequin.Checked then begin
      Append(ArlequinBatch);
      Individ := 1;
      FileCounter := DirOut + '/Arlequin/sim_' + IntToStr(i) + '.arp';
      Writeln(ArlequinBatch, FileCounter);
      CloseFile(ArlequinBatch);
      AssignFile(ArlequinProject, FileCounter);
      Rewrite(ArlequinProject);
      Writeln(ArlequinProject, '[Profile]');
      Writeln(ArlequinProject, '   Title="DNA sequence data generated using
coalescent simulations"');
      Writeln(ArlequinProject, '   NbSamples=1');
      Writeln(ArlequinProject, '   GenotypicData=1');
      Writeln(ArlequinProject, '   DataType=DNA');
      Writeln(ArlequinProject, '   LocusSeparator=NONE');
      Writeln(ArlequinProject, '[Data]');
      Writeln(ArlequinProject, ' [[Samples]]');
      Writeln(ArlequinProject, '   SampleName="Homogenous population"');
      ArlequinSampleSize := IntToStr(k div 2);
      Writeln(ArlequinProject, '   SampleSize=' + ArlequinSampleSize);
      Writeln(ArlequinProject, '   SampleData= {');

      for ai := 0 to k-1 do begin
        RandomInd := RandomChosen[ai];
        if (Odd(ai) = False) and (ai <> 0) then
          Individ := Individ+1;
        if (odd(ai) = False) or (ai = 0) then begin
          StringName := IntToStr(Individ) + ' ';
          Write(ArlequinProject, StringName: 10);
          Write(ArlequinProject, ' 1 ');
        end
        else begin
          Write(ArlequinProject, '            ');
        end;
        for aj := 1 to SeqLength do begin
          valueChar := Data[aj-1, RandomInd-1];
          Write(ArlequinProject, UpperCase(valueChar));
        end;
        Writeln(ArlequinProject);
      end;
      Writeln(ArlequinProject, '}');
      CloseFile(ArlequinProject);
```

```
  end;
 end

 else begin
  If rdgOutput.Items[rdgOutput.ItemIndex] = 'Fasta' then begin
   Append(outfileSeq);
   Writeln(outfileSeq, IntToStr(k) + ' ' + IntToStr(SeqLength));
   for v := 0 to k-1 do begin
    StringName := IntToStr(v+1) + '_' + IntToStr(i) + ' ';
    Write(outfileSeq, StringName:10);
    for w := 0 to SeqLength-1 do begin
     base := Data[w,v];
     Write(outfileSeq, UpperCase(base));
    end;
    Writeln(outfileSeq);
   end;
   WriteLn(outfileSeq);
   CloseFile(outfileSeq);
  end;

  If rdgOutput.Items[rdgOutput.ItemIndex] = 'Clustal' then begin
   Append(outfileSeq);
   for v := 0 to k-1 do begin
    Writeln(outfileSeq, '>' + IntToStr(v+1)+'_'+IntToStr(i));
    for w := 0 to SeqLength-1 do begin
     base := Data[w,v];
     Write(outfileSeq, UpperCase(base));
    end;
    Writeln(outfileSeq);
   end;
   WriteLn(outfileSeq);
   CloseFile(outfileSeq);
  end;

  If rdgOutput.Items[rdgOutput.ItemIndex] = 'MrBayes' then begin
   Append(outfileSeq);
   Writeln(outfileSeq,'#NEXUS');
   Writeln(outfileSeq);
   Writeln(outfileSeq,'BEGIN DATA;');
   Writeln(outfileSeq, 'DIMENSIONS ' + 'NTAX=' + IntToStr(k)+ ' NCHAR='
+ IntToStr(SeqLength) + ';');
   Writeln(outfileSeq, 'FORMAT DATATYPE=DNA;');
   Writeln(outfileSeq, 'MATRIX');
   for v := 0 to k-1 do begin
    Writeln(outfileSeq, IntToStr(v+1)+'_'+IntToStr(i));
    for w := 0 to SeqLength-1 do begin
     base := Data[w,v];
     Write(outfileSeq, UpperCase(base));
    end;
    Writeln(outfileSeq);
```

```
            end;
            Writeln(outfileSeq, ';');
            Writeln(outfileSeq, 'end;');
            WriteLn(outfileSeq);
            Writeln(outfileSeq, 'begin mrbayes;');
            Writeln(outfileSeq, 'set autoclose=yes;');
            if radJC69.Checked then
              Nst := 1
            else Nst := 2;
            if chkShapeParameter.Checked then
              StringShape := 'rates=gamma'
            else StringShape := '';
            Writeln(outfileSeq, 'lset nst=' + IntToStr(Nst) + ' ' + StringShape + ';');
            Writeln(outfileSeq, 'mcmc ngen=1000 printfreq=100 samplefreq=10
nchains=4 savebrlens=yes;');
            Writeln(outfileSeq, 'end;');
            CloseFile(outfileSeq);
          end;

          If rdgOutput.Items[rdgOutput.ItemIndex] = 'Nexus' then begin
            Append(outfileSeq);
            Writeln(outfileSeq,'#NEXUS');
            Writeln(outfileSeq);
            Writeln(outfileSeq,'BEGIN DATA;');
            Writeln(outfileSeq, 'DIMENSIONS ' + 'NTAX=' + IntToStr(k)+ ' NCHAR='
+ IntToStr(SeqLength) + ';');
            Writeln(outfileSeq, 'FORMAT DATATYPE=DNA;');
            Writeln(outfileSeq, 'MATRIX');
            for v := 0 to k-1 do begin
              Writeln(outfileSeq, IntToStr(v+1)+'_'+IntToStr(i));
              for w := 0 to SeqLength-1 do begin
                base := Data[w,v];
                Write(outfileSeq, UpperCase(base));
              end;
              Writeln(outfileSeq);
            end;
            Writeln(outfileSeq, ';');
            Writeln(outfileSeq, 'end;');
            WriteLn(outfileSeq);
            CloseFile(outfileSeq);
          end;

          If chkArlequin.Checked then begin
            Append(ArlequinBatch);
            FileCounter := DirOut + '/Arlequin/sim_' + IntToStr(i) + '.arp';
            Writeln(ArlequinBatch, FileCounter);
            CloseFile(ArlequinBatch);
            AssignFile(ArlequinProject, FileCounter);
            Rewrite(ArlequinProject);
            Writeln(ArlequinProject, '[Profile]');
```

```
          Writeln(ArlequinProject, '   Title="DNA sequence data generated using
coalescent simulations"');
          Writeln(ArlequinProject, '   NbSamples=1');
          Writeln(ArlequinProject, '   GenotypicData=0');
          Writeln(ArlequinProject, '   DataType=DNA');
          Writeln(ArlequinProject, '   LocusSeparator=NONE');
          Writeln(ArlequinProject, '[Data]');
          Writeln(ArlequinProject, ' [[Samples]]');
          Writeln(ArlequinProject, '   SampleName="Panmictic population"');
          ArlequinSampleSize := IntToStr(k);
          Writeln(ArlequinProject, '   SampleSize=' + ArlequinSampleSize);
          Writeln(ArlequinProject, '   SampleData= {');

          for ai := 0 to k-1 do begin
            StringName := IntToStr(ai+1) + ' ';
            Write(ArlequinProject, StringName:10);
            Write(ArlequinProject, ' 1 ');
            for aj := 1 to SeqLength do begin
              valueChar := Data[aj-1, ai];
              Write(ArlequinProject, UpperCase(valueChar));
            end;
            Writeln(ArlequinProject);
          end;
          Writeln(ArlequinProject, '}');
          CloseFile(ArlequinProject);
        end;

      end;


    end;

    value := TreeMatrix[k-2, 0];
    Tmrca[i-1] := value;
    InfAlleles[i-1] := SegSites;
    FinAlleles[i-1] := TotalSegSitesFin;

    Append(TreeMatrixFile);
    Writeln(TreeMatrixFile, '#' + IntToStr(i));
    for tmf2 := 0 to k-2 do begin
      for tmf := 0 to 2 do begin
        value := TreeMatrix[tmf2, tmf];
        Write(TreeMatrixFile, IntToStr(value) + ' ');
      end;
      Writeln(TreeMatrixFile);
    end;
    CloseFile(TreeMatrixFile);

    Append(MutationMatrixFile);
    Writeln(MutationMatrixFile, '#' + IntToStr(i));
```

```
        for mmf := 0 to CountMutation-1 do begin
          for mmf2 := 0 to 1 do begin
            value := MutationMatrix[mmf,mmf2];
            Write(MutationMatrixFile, IntToStr(value) + ' ');
          end;
          Writeln(MutationMatrixFile);
        end;
        CloseFile(MutationMatrixFile);
        NumberMutationArray[i-1] := CountMutation;


      end; // end of for i:=1 to NumberSimulations loop
      Outputs(NumberSimulations,0,k);
      TreeCounter := NumberSimulations;
      DrawTree;
    end;  // end of sequence model loop
    EndTime := Time;
    LogFile(k,N,NBeforeGrowth,SeqLength,GrowthStart,GrowthEnd,
NumberSimulations, StartTime, EndTime, MuRate,FreqA,FreqC,FreqG,FreqT);

    SimDone := True;
  end  //end of if (k>0) and (N>0) loop
  else begin
    resultString := 'Please insert Sample and Population Sizes';
    Application.MessageBox(PChar(resultString), 'WARNING', [smbok]);
  end;

end;

procedure TfrmRandCoal.SaveGenealogy(Sender: TObject);

var
 resultString : string;

begin
  if (SimDone = False) then begin
    resultString := 'Please run simulations first';
    Application.MessageBox(PChar(resultString), 'WARNING', [smbok]);
  end
  else begin
    dlgSaveFile.Execute;
    Image1.Picture.Bitmap.SaveToFile(dlgSaveFile.FileName);
  end;
end;

procedure TfrmRandCoal.CreateForm(Sender: TObject);

begin
  Image3.Picture.LoadFromFile('./img/meep_logo.png');
  SimDone := False;
```

```
  DirOut := '.';
end;

procedure TfrmRandCoal.AssignDirectory(Sender: TObject);

begin
  MemDir.Clear;
  DirOut := DirectoryTreeView1.Directory;
  MemDir.Lines.Add(DirOut);
end;

procedure TfrmRandCoal.TreeFwd(Sender: TObject);

var
  TestString : string;
  ReadString : string;
  ReadChar : string;
  ReadInt : integer;
  i,j : integer;
  k,code : integer;
  resultString : string;
  NLoci : integer;
  SimNumber, LocusNumber : integer;

begin
  if (SimDone = False) then begin
    resultString := 'Please run simulations first';
    Application.MessageBox(PChar(resultString), 'WARNING', [smbok]);
  end
  else begin
    Val(edtSampleSize.Text, k, code);
    Val(edtNumLoci.Text, NLoci, code);
    if chkMSats.Checked then begin
      if TreeCounter = NumberSimulations*NLoci then
        TreeCounter := 1
      else
        TreeCounter := TreeCounter + 1;
      AssignFile(TreeMatrixFile, DirOut + '\' + 'TreeMatFile.txt');
      Reset(TreeMatrixFile);
      AssignFile(MutationMatrixFile, DirOut + '\' + 'MutMatFile.txt');
      Reset(MutationMatrixFile);
      TestString := '#' + IntToStr(TreeCounter);

      repeat
        Read(TreeMatrixFile, ReadString);
        if (ReadString = TestString) then begin

          for i := 0 to k-2 do begin
            for j := 0 to 2 do begin
              Read(TreeMatrixFile, ReadInt);
```

```
            TreeMatrix[i,j] := ReadInt;
          end;
          Readln(TreeMatrixFile);
        end;
      end
      else
        Readln(TreeMatrixFile);
    until (ReadString = TestString);

    MutationMatrix := nil;
    CountMutation := NumberMutationArray[TreeCounter-1];
    SetLength(MutationMatrix, CountMutation, 2);
    repeat
      Read(MutationMatrixFile, ReadString);
      if (ReadString = TestString) then begin

        for i := 0 to CountMutation-1 do begin
          for j := 0 to 1 do begin
            Read(MutationMatrixFile, ReadInt);
            MutationMatrix[i,j] := ReadInt;
          end;
          Readln(MutationMatrixFile);
        end;
      end
      else
        Readln(MutationMatrixFile);
    until (ReadString = TestString);

    DrawTree;

    LocusNumber := TreeCounter mod NLoci;
    SimNumber := Ceil(TreeCounter/NLoci);
    if LocusNumber = 0 then
      LocusNumber := TreeCounter div SimNumber;

    with Image1,Canvas do begin
      Font.Color := clBlack;
      Font.Size := 10;
      Textout(10, 510, 'Microsatellite Locus ' + IntToStr(LocusNumber) + ' Simulation
' + IntToStr(SimNumber));
    end;
  end

  else begin
    if TreeCounter = NumberSimulations then
      TreeCounter := 1
    else
    TreeCounter := TreeCounter + 1;
    AssignFile(TreeMatrixFile, DirOut + '\' + 'TreeMatFile.txt');
    Reset(TreeMatrixFile);
```

```
    AssignFile(MutationMatrixFile, DirOut + '\' + 'MutMatFile.txt');
    Reset(MutationMatrixFile);
    TestString := '#' + IntToStr(TreeCounter);

    repeat
      Read(TreeMatrixFile, ReadString);
      if (ReadString = TestString) then begin

        for i := 0 to k-2 do begin
          for j := 0 to 2 do begin
            Read(TreeMatrixFile, ReadInt);
            TreeMatrix[i,j] := ReadInt;
          end;
          Readln(TreeMatrixFile);
        end;
      end
      else
        Readln(TreeMatrixFile);
    until (ReadString = TestString);

    MutationMatrix := nil;
    CountMutation := NumberMutationArray[TreeCounter-1];
    SetLength(MutationMatrix, CountMutation, 2);
    repeat
      Read(MutationMatrixFile, ReadString);
      if (ReadString = TestString) then begin

        for i := 0 to CountMutation-1 do begin
          for j := 0 to 1 do begin
            Read(MutationMatrixFile, ReadInt);
            MutationMatrix[i,j] := ReadInt;
          end;
          Readln(MutationMatrixFile);
        end;
      end
      else
        Readln(MutationMatrixFile);
    until (ReadString = TestString);

    DrawTree;
    with Image1,Canvas do begin
      Font.Color := clBlack;
      Font.Size := 10;
      Textout(10, 510, IntToStr(TreeCounter) + '\' + IntToStr(NumberSimulations));
    end;
  end;
 end;
 CloseFile(TreeMatrixFile);
 CloseFile(MutationMatrixFile);
end;
```

```
procedure TfrmRandCoal.TreeBack(Sender: TObject);

var
  TestString : string;
  ReadString : string;
  ReadInt : integer;
  i,j,k,code : integer;
  resultString : string;
  NLoci : integer;
  LocusNumber, SimNumber : integer;


begin
  if (SimDone = False) then begin
    resultString := 'Please run simulations first';
    Application.MessageBox(PChar(resultString), 'WARNING', [smbok]);
  end
  else begin
    Val(edtSampleSize.Text, k, code);
    Val(edtNumLoci.Text, NLoci, code);
    if chkMSats.Checked then begin
      if TreeCounter = 1 then
        TreeCounter := NumberSimulations*NLoci
      else
        TreeCounter := TreeCounter - 1;
      AssignFile(TreeMatrixFile, DirOut + '\' + 'TreeMatFile.txt');
      Reset(TreeMatrixFile);
      AssignFile(MutationMatrixFile, DirOut + '\' + 'MutMatFile.txt');
      Reset(MutationMatrixFile);
      TestString := '#' + IntToStr(TreeCounter);

      repeat
        Read(TreeMatrixFile, ReadString);
        if (ReadString = TestString) then begin

          for i := 0 to k-2 do begin
            for j := 0 to 2 do begin
              Read(TreeMatrixFile, ReadInt);
              TreeMatrix[i,j] := ReadInt;
            end;
            Readln(TreeMatrixFile);
          end;
        end
        else
          Readln(TreeMatrixFile);
      until (ReadString = TestString);

      MutationMatrix := nil;
      CountMutation := NumberMutationArray[TreeCounter-1];
```

```
    SetLength(MutationMatrix, CountMutation, 2);
    repeat
      Read(MutationMatrixFile, ReadString);
      if (ReadString = TestString) then begin

        for i := 0 to CountMutation-1 do begin
          for j := 0 to 1 do begin
            Read(MutationMatrixFile, ReadInt);
            MutationMatrix[i,j] := ReadInt;
          end;
          Readln(MutationMatrixFile);
        end;
      end
      else
        Readln(MutationMatrixFile);
    until (ReadString = TestString);

    DrawTree;

    LocusNumber := TreeCounter mod NLoci;
    SimNumber := Ceil(TreeCounter/NLoci);
    if LocusNumber = 0 then
      LocusNumber := TreeCounter div SimNumber;


    with Image1,Canvas do begin
      Font.Color := clBlack;
      Font.Size := 10;
      Textout(10, 510, 'Microsatellite Locus ' + IntToStr(LocusNumber) + ' Simulation
' + IntToStr(SimNumber));
    end;
  end

  else begin
    if TreeCounter = 1 then
      TreeCounter := NumberSimulations
    else
    TreeCounter := TreeCounter - 1;
    AssignFile(TreeMatrixFile, DirOut + '\' + 'TreeMatFile.txt');
    Reset(TreeMatrixFile);
    AssignFile(MutationMatrixFile, DirOut + '\' + 'MutMatFile.txt');
    Reset(MutationMatrixFile);
    TestString := '#' + IntToStr(TreeCounter);

    repeat
      Read(TreeMatrixFile, ReadString);
      if (ReadString = TestString) then begin

        for i := 0 to k-2 do begin
          for j := 0 to 2 do begin
```

```
            Read(TreeMatrixFile, ReadInt);
            TreeMatrix[i,j] := ReadInt;
          end;
          Readln(TreeMatrixFile);
        end;
      end
      else
        Readln(TreeMatrixFile);
    until (ReadString = TestString);

    MutationMatrix := nil;
    CountMutation := NumberMutationArray[TreeCounter-1];
    SetLength(MutationMatrix, CountMutation, 2);
    repeat
      Read(MutationMatrixFile, ReadString);
      if (ReadString = TestString) then begin

        for i := 0 to CountMutation-1 do begin
          for j := 0 to 1 do begin
            Read(MutationMatrixFile, ReadInt);
            MutationMatrix[i,j] := ReadInt;
          end;
          Readln(MutationMatrixFile);
        end;
      end
      else
        Readln(MutationMatrixFile);
    until (ReadString = TestString);

    DrawTree;
    with Image1,Canvas do begin
      Font.Color := clBlack;
      Font.Size := 10;
      Textout(10, 510, IntToStr(TreeCounter) + '\' + IntToStr(NumberSimulations));
    end;
  end;
 end;
 CloseFile(TreeMatrixFile);
 CloseFile(MutationMatrixFile);
end;

end.
```