Date of acceptance Grade

Instructor

Improving the throughput of the forward population genetic simulation environment simuPOP

Juhana Kammonen

Helsinki November 6, 2013

MSc thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Juhana Kammonen

Työn nimi — Arbetets titel — Title

Improving the throughput of the forward population genetic simulation environment simuPOP

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	November 6, 2013	49 pages

Tiivistelmä — Referat — Abstract

Biological populations arise, develop and evolve under a series of well-studied laws and fairly regular mechanisms. Population genetics is a field of science, that aims to study and model these laws and the genetic composition and diversity of populations of various types of species and life. At best, population genetic models can be of use in verifying past events of a population and eventually reconstructing unknown population histories in light of multidisciplinary evidence. An example case of this is the research concerning human population prehistory of Finland.

Population simulations are a sub-branch of the rapidly developing field of bioinformatics and can be divided into two pipelines: forward-in-time and backward-in-time (coalescent). The methodologies enable *in silico* testing of the development of genetic composition of individuals in a well-defined population. This thesis focuses on the forward-in-time approach. Multiple pieces of software exist today for forward population simulations, and `simuPOP` [http://simupop.sourceforge.net] probably is the single most flexible one of them. Being able to incorporate transmission of genomes and arbitrary individual information between generations, `simuPOP` has potential applications even beyond population genetics. However, `simuPOP` tends to use an enormous amount of computer random access memory when simulating large population sizes.

This thesis introduces three approaches to improve the throughput of `simuPOP`. These are *i)* introducing scripting guidelines, *ii)* approximating a complex simulation using the inbuilt biallelic mode of `simuPOP` and *iii)* changes in the source code of `simuPOP` that would enable improved throughput. A previous `simuPOP` script designed to simulate past demographic events of Finnish population history is used as an example. A batch of 100 simulation runs is run on three versions of the previous script: standard, modified and biallelic. As compared to the standard mode, the modified simulation script performs marginally faster. Despite doubling the user time of a single simulation run, the biallelic approximation method proves to consume three times less random access memory still being compatible from the population genetic point of view. This suggests that built-in support for the biallelic approximation could be a valuable supplement to `simuPOP`.

Evidently, `simuPOP` can be applied to very complex forward population simulations. The use of individual information fields enables the user to set up arbitrary simulation scenarios. Data structure changes at source code level are likely to improve throughput even further. Besides introducing improvements and guidelines to the simulation workflow, this thesis is a standalone case study concerning the use and development of a bioinformatics software. Furthermore, an individual development version of `simuPOP` called `simuPOP-rev` is founded with the goal of implementing the source code changes suggested in this thesis.

ACM Computing Classification System (CCS): D.1 [Programming techniques], G.1.6 [Optimization], H.3 [Information storage and retrieval]

Avainsanat — Nyckelord — Keywords
simuPOP, memory allocation, data structures, population simulations, Finnish population history

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Bioinformatics in a very broad sense could be described as the science of processing data from biological objects, such as DNA or amino acid sequences. The structure and chemical properties of DNA were discovered already in the 1950's [WC1953]. Further studies eventually developed into the *central dogma of molecular biology* describing how the information within DNA is processed leading from transcription into a ready protein. Bioinformatics saw a significant rise during the 1990's together with the development of ever improving practical methods for obtaining biological data from model species, especially next-generation sequencing. More recently, bioinformatics has highly benefited from the development of high-throughput computers. Bioinformatics has several sub-branches and this thesis briefly introduces some main ones of them: population genetics, DNA sequencing, systems biology and genomics. Special emphasis of this thesis is on population genetics, more accurately computer-aided population simulation methods, while other topics such as RNA sequencing, proteomics or microbial ecology bioinformatics are left out of the scope of this study.

Computer simulations and modelling are used extensively in bioinformatics, at least in the sub-branches of population genetics and systems biology. Generally speaking, computer simulations can be utilized on almost any field of science to test different empirical hypotheses or to look for potential solutions to different problems. While being unable to provide absolute proof for the validity of a specific simulation scenario, simulation approaches often are a convenient, cost-effective and repeatable method for distinguishing between likely and unlikely scenarios. Great care should be put on whether a simulation model is valid and worth the resources invested to the development of the model [Sar2005].

Population genetic computer simulation refers to modelling a part of a genome (or sometimes the entire genome) of a studied species with a simulation software based on well-defined mathematical and statistical models. The software applied should implement at least basic population genetic features such as populations, individuals with genomes, demographics, migration between populations and mutation. Considering population genetic simulations, two main approaches are the coalescent (backward-in-time) method and the forward-in-time method. The vast domain of simulation software currently available can be roughly divided into the two abovementioned categories. Along with the sheer quantity of simulation software available, different input and output file formats are even more abundant. There is a clear

need for a common, more general file format to unify the shattered field of data input and output [Exc2006]. While this thesis does not directly aim to improve the situation, unified data format is one of the major future aspects of simulation software developers. More generally, population genetic simulation methods have provided both academic and everyday users a way to simulate population histories where they would be unreachable by other means. Especially prehistories of populations, meaning the completely undocumented portions of the population histories, become tractable with this methodology.

A population genetic simulation environment called `simuPOP` [Pen2005] has proven to be a flexible and well-maintained simulation tool for simulating human population history scenarios in Finland [Heg2010, Sun2010, Sun2013]. Research concerning the prehistory of Finland is a good example case of applying multidisciplinary evidence to get a wider perspective on the study subject.

Basic population genetic features, such as genomes, individuals, populations, migration and mutation have been implemented in `simuPOP`, and it is possible to simulate populations of any living organism. In essence, `simuPOP` is a collection of Python programming language [Ros2011] modules that users import into their simulation program script. On one hand, consequences are that a fairly confident level of skill in Python programming from users is required. On the other hand, free use of Python enables the user to incorporate arbitrary mathematical models into the simulation run limited only by the application programming interface of the language. Evidently, `simuPOP` is the most versatile population simulation tool available, at least when concerning forward population simulations (see Section 2.3.2). A review of various population simulation approaches and tools is given in Section 2.3.

There are three main objectives in this thesis: i) to introduce scripting guidelines for efficient use of `simuPOP`, ii) investigate the feasibility of using `simuPOP`'s bialleic mode as a viable approximation for the standard multi-allelic mode, and iii) to introduce changes in the source code of `simuPOP` to improve memory-loading especially in the multi-allelic mode. To achieve the latter, an independent development version of `simuPOP` called `simuPOP-rev` is introduced. Moreover, as a Finnish population genetic scenario is used as an example, the simulation results are evaluated in the light of present-day understanding of the genetic composition of Finnish human population.

This thesis is structured as follows: Section 2 will review a selected set of literature concerning bioinformatics in general as well as population genetics and population

genetic simulation methods. Previous simulation approaches concerning population history of Finland are also presented. Section 3 will introduce in detail the methods and materials used in this thesis in order to achieve the abovementioned objectives. Section 4 will present the main results of the applied methods. Finally, the results of this thesis and future work concerning the presented topics are discussed in detail in Section 5.

2 Literature review

It had been imagined already in the first half of the 1900's that a chemical compound called deoxyribonucleic acid (DNA) is the physical entity that encodes biological information. James Watson and Francis Crick then together with Rosalind Franklin discovered the structure of DNA from X-ray diffraction images in 1953 [WC1953]. The study of this structure and its behavior within the cells eventually led to the development of the framework of protein synthesis, i.e. that the biological information within DNA is used as a recipe for a protein through processes of transcription into ribonucleic acid (RNA) and translation of it. The information is encoded in the sequential order of the four nucleotide bases: adenine (A), cytosine (C), guanine (G) and thymine (T). Thymine is replaced by uracil (U) in the case of RNA. Systematic (usually computer-aided) storage, retrieving and analysis of the information stored in DNA and its derivatives is at the heart of bioinformatics, which is a rather novel field of computational science. The framework of protein synthesis is commonly known as the *central dogma of molecular biology* [Cri1970, Sha2009].

Bioinformatics applications started blooming as quickly developing DNA sequence extraction and storage methods began overwhelming the computational methods of the time for analyzing the data. Bioinformaticians can aid the increased need for efficient data analysis techniques by designing better methods and algorithms for processing the abundant data. These methods often are incorporated into bioinformatics software. The field of different applications and file formats in bioinformatics appears abundant, complex and shattered [Exc2006].

In living organisms, all species tend to form populations mainly for better chances of producing offspring and survival. It is of interest to study populations to understand their dynamics. Regarding population simulations, various simulation software are available to users. A forward population simulation environment called `simuPOP` [Pen2005] has gained publicity during recent years. The software is under active

development and new release versions are published as often as every 5 months.

The computational developments presented in this thesis are introduced alongside a brief look at the human population history of Finland, a suitable case study. How ancient Finland was actually populated after the after the Vistula Ice Age remains a question of debate. Evidence from archaeology indicates that there has been human activity for at least 11,000 years beginning almost immediately after the retreat of the glacial ice [Pes2010, Tal2010]. Moreover, there has been debate as to whether hominid inhabitation existed already before the Vistula Ice Age, more than 100,000 years ago. These inhabitants would have had to be ancestors of modern humans, most likely Neanderthals (*Homo neanderthalensis*). This rather bold assumption may be supported by latest evidence from *Susiluola* ([Mag1998, Pur2004]) (literally translated as *wolf cave*) in Karijoki, western Finland.

The forward population genetic simulation environment `simuPOP` has been implemented using programming language C++ [Str2000] and has an open source code. While the contribution of the main developers has concentrated on adding new functionalities based on user demand, less attention has been paid to technical issues such as memory efficiency and data structures. With an open source code, program developers (and any computer user for that matter) have access to the code. As the code is distributed under a General Public License (GPL), developers are allowed to distribute individual and modified versions of the software on the condition that they endorse the GPL license. The author of this thesis has done exactly this and started an individual version of the software called `simuPOP-rev` in order to control the source code level changes suggested in this thesis.

2.1 Bioinformatics - a rapidly developing field of science

Bioinformatics as a science is rather novel. The field saw its rise as information produced by quickly developing DNA sequencing methods and the extensive deposition of new biological sequence data into various databases began overwhelming the methods for efficiently analyzing the data [Tui2005]. Bioinformaticians can aid the need for efficient data analysis techniques by providing algorithms for processing the incredibly abundant data. There are many sub-branches within what can be called bioinformatics. This section briefly presents some main ones of these.

Population genetics is the study of genetic properties of populations, such as population size and genetic diversity within a population. These properties can be

studied by taking genetic samples from populations and analyzing e.g. haplotype content and diversity in the population. Population simulation tools are a cost-efficient approach in case repeated sampling and analysis is needed. This sub-branch of bioinformatics along with different simulation methods will be discussed in greater detail in Section 2.2.

DNA sequencing is a field specialized in production, application and use of devices that are capable of approximating an enumeration of the partial or whole genome of a species. For example, the human (*Homo sapiens*) genome contains more than three billion base pairs. With DNA sequencing methods, it has been possible to roughly sort all these base pairs into a single reference genome. This is the case for the genomes of many other species too. DNA sequencing theory constitutes the fundamental ideas and problems of DNA sequencing. Bioinformaticians typically develop and apply algorithms that solve these problems as efficiently as possible. Along with ever developing sequencing methods, the development of algorithms and computer processor technology will allow for sequencing of the most complex genomes.

Systems biology is concerned with understanding and modelling biological systems such as different pathways in metabolism or disease developments. The ambitious goal of systems biology is the complete definition and modelling of all possible biological systems. Moreover, efficient algorithms are essential for modelling the virtually unlimited number of biological systems.

Genomics is a discipline in genetics, though closely related to bioinformatics, and is concerned with the study of the function and structure of arbitrary genomes. Various databases are maintained to enable continuous updates and integration of new data.

To sum up, bioinformatics is a field of science that brings together computer scientists, biologists and experts from other fields to join the multidisciplinary effort of studying, modelling and understanding different structures and mechanisms of life. Unfortunately, the scope of a Master's thesis cannot be broad enough to present each individual sub-branch of bioinformatics in detail. An extensive outlook (in Finnish) into core bioinformatics methods has been published by the Finnish IT Center for Science (CSC) [Tui2005].

2.2 Population genetics

Population genetics developed alongside with the discovery of DNA as the physical natural entity that encodes biological information. The profile of inherited material within a population is strongly varied depending on the living conditions of a species. Through population genetic analyses it has become clear that properties such as effective and actual population sizes, individual fitness and subpopulation division have a significant effect on the genetic diversity of populations.

Population bottleneck is one of the key concepts of population genetics. In a narrow sense, a population bottleneck means that the number of individuals in a population becomes very small and remains there for some time. In a more broad sense, defining exactly how small the population should be or how long should the population size remain at that number to actually consitute a bottleneck is required when using the concept. This usually depends on the species and the population's interactions with the environment. Population bottlenecks are usually preceded by a distinct decline in population size and are followed by either population expansion or, in some cases, extinction. An example illustration of population bottlenecks is given in Figure 1.

Founder effect actually is an instance of a population bottleneck and basically means the genetic effect that the small number of individuals can have in a founding population. A founder effect is illustrated in Figure 1.

Genetic drift is another population genetic phenomenon that often has a dramatic effect on the genetic composition of a population, especially if the population is recently founded. Despite being initially very diverse in different genes, after a relatively small amount of generations have passed, the population may have lost more than half of its initial diversity. This effect is enhanced if the population is isolated and there is no gene flow or migration into the population from outside. Genetic drift often occurs in demographic events known as *range expansions* [Ray2009]. The first phase of a range expansion essentially is a founder effect.

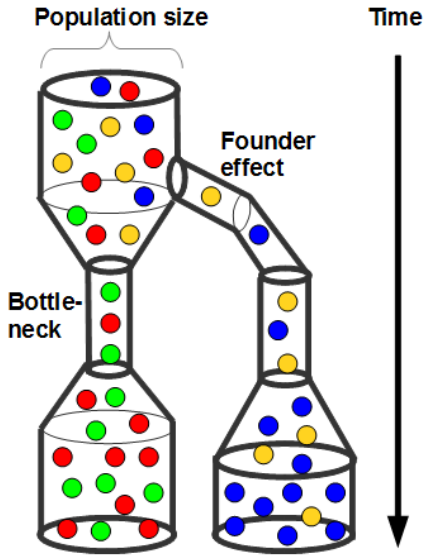

Figure 1: A simplified example population model illustrating population bottlenecks and a founder effect. The width of the cylinder represents population size at a given time. Coloured spheres represent individuals with a specific allele indicated by the color of the sphere. In this specific example, genetic drift causes the yellow allele to become almost extinct in the final population. Migration between the split populations is likely to occur - at least in case of human populations.

2.3 Population genetic simulation approaches and tools

Computer simulations are used in various fields of science mostly for their great cost-efficiency compared to other methods. Consider, for example, a genetic study where authentic genetic data is gathered by taking biological samples. If samples become destroyed or otherwise affected by, for example, contaminated instruments (not unfamiliar in laboratories) [Gra1996, Doi2004], one will need to collect a clean set of new samples. In case the damage is irreparable, the cost of collecting samples and processing them in laboratory conditions could be intolerable and funding for repeated sampling may not always be available.

There are many population simulators available today: coalescent simulators like `simcoal` and `BayesSSC` [Exc2000] (http://cmpg.unibe.ch/software/simcoal) and forward simulators like IM [Nie2001] or EASYPOP [Bal2001] to mention a few. The main focus of this thesis is on a forward population simulator `simuPOP` [Pen2005] and multiple ways to improve the throughput of this simulator are introduced. `simuPOP`

proves to be a very flexible forward population simulator and, in case sufficient computing power is available, allows for arbitrarily complex simulations without the need for source code changes present in many other software [Pen2005].

2.3.1 Coalescent simulators

Coalescent simulators are population simulators based on the serial coalescent and are very often based on Bayesian methods. This is usually implemented in the form of approximate Bayesian computing (ABC) [Bea2002]. The common feature of coalescent simulators is that the simulation proceeds *backwards in time*, beginning from present day and then proceeding generation by generation back towards the common ancestor of a sample. The Bayesian methodology is applied when prior probability distributions are assigned to population genetic events such as gradients in population sizes or time points of founder effects. Moreover, models of evolutionary change (mutation) are applied separately to the entire simulated genealogy in some instances [Exc2000].

2.3.2 Forward simulators

Forward simulators provide a rather different approach to population simulations. In forward simulation, a starting population is usually initialized and a starting time point set. The simulation then proceeds *forward-in-time*, hence the name *forward simulation*, until some pre-defined endpoint or other conditions for ending the simulation are met. Various statistics of interest can be tracked during the simulation, such as population size, migration and mutation events. Forward simulations are very useful for sorting out the most likely scenarios for the history of a population. For this thesis, `simuPOP` [Pen2005] serves as a prime example of a forward simulator.

`simuPOP` is used through a programming (scripting) language called Python [Ros2011]. Thanks to this feature, the user can set up different forward simulation scenarios ranging from very simple to immensely complex, theoretically limited only by the application programming inteface of Python. The simulated species can be anything from bacteria to plants to human, as the genomes of the individuals are defined in detail before the start of the simulation. A flowchart illustrating the workflow of `simuPOP` is shown in Figure 2.

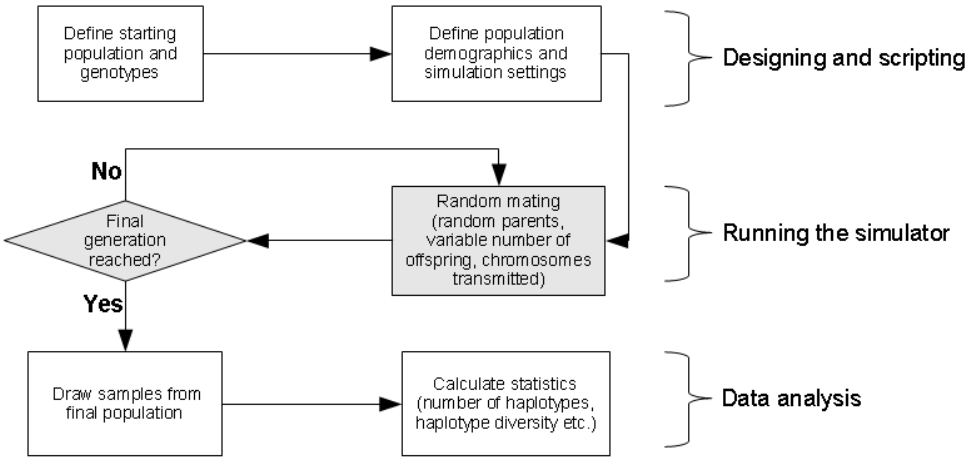

Figure 2: A flowchart illustrating the basic workflow of simuPOP. Image: Sundell *et al.* 2010 [Sun2010].

2.3.3 Summarizing statistics of interest

Tools for calculating various summary statistics of interest are usually built-in in population simulators. In the case of forward simulators, these statistics of interest are, for instance, population size at time t, number of haplotypes in the population and genetic diversity of the population. In case autosomes are also simulated, it is possible to calculate a statistic known as linkage disequilibrium (LD).

Population size at time t is one of the very basic statistics of population genetics. It is the absolute number of individuals in a population. The effects of population size to the genetic diversity of a population can be drastic, especially if the size remains very small for prolonged periods of time.

Number of haplotypes in a population or a subpopulation is descriptive of the individual level genetic diversity in the population. The closer the number of haplotypes in the population is to the actual population size, the more diverse the population is. This statistic can be easily calculated in `simuPOP` and other population simulators. The statistic for the number of haplotypes is noted in this thesis by letter A, indicating the same statistic as in a study of actual Finnish genetic data

[Pal2009].

Genetic diversity (also *haplotype diversity*) of a population or subpopulation is indicative of the similarity of individuals within the population. There are many ways of computing genetic diversity, one of the most common being the formula by Masatoshi Nei [Nei1973], in which the genetic diversity $\hat{H}$ of a population is defined as:

$$\hat{H} = n(1 - \sum_{i=1}^{a} x_i)/(n-1) \; , \tag{1}$$

where n is the number of individuals in the population, a is the total number of haplotypes in the population and x_i is the frequency of the i-th haplotype. This formula also is known as *Nei's gene diversity*. Whenever using the concept *genetic diversity* in this thesis, the mathematical rationale within will be defined by this formula.

Effective population size of a population is a distinct measure the definition of which depends strongly on the application. It describes the number of individuals in a population that are actively contributing to the transmission of genomes from their generation to their offspring by reproduction. Typically, this size (noted as N_e) is much smaller than the actual population size N. A good review of the various factors affecting the definition of the statistic is given in a recent publication [Cha2009]. In this thesis, effective population size N_e refers to the statistic defined in an evolutionary human genetics source book [Job2013]:

$$N_e = \frac{1}{\frac{1}{T}\sum_{i=1}^{T}\frac{1}{N_i}} \; , \tag{2}$$

where N_i is the actual number of individuals in the i-th subpopulation and T is the total number of subpopulations. This means that, as population sizes never remain constant, N_e is the *harmonic mean* of subpopulation sizes as these population sizes vary through generations. Moreover, it is intuitive that the statistic tends to be dominated by a subpopulation that enters a narrow bottleneck.

Linkage disequilibrium is a population statistic calculated from autosomes. It portrays the disparity of random and non-random linkage between alleles. Autosomes were not simulated in the original simulation studies [Sun2010, Sun2013] and also were not simulated in any of the simulations presented in this thesis.

Hardy-Weinberg equilibrium is a theorem that states that frequencies of alleles

in a population should remain constant from generation to generation in case no evolutionary influences, such as mating, selection , mutation, gene flow or genetic drift, are affecting the population.

F-ST also known as the fixation index in a population is one of the so called *F-statistics* that describe the deviation of the observed population's genetic profile as compared to the expected profile (essentially the Hardy-Weinberg equilibrium). Computation of the statistic usually involves a series of pairwise comparisons between subpopulations. If F-ST between two subpopulations is small, the allele frequencies between the subpopulations are close to one another. The statistic may also be calculated for the total population. A detailed introduction to the statistic is given in a recent survey article [Hol2009].

Despite all the statistics above can be monitored during a simulation, also in `simuPOP`, the test simulations in this thesis make use of the three former statistics (population size, number of haplotypes and genetic diversity) due to the following facts. First, the original studies [Heg2010, Sun2010, Sun2013] to which this thesis compares its results used the same statistics. Effective population size N_e was computed in the results of the most recent of the studies [Sun2013] but only after the actual simulation. Second, autosomes were not simulated in the original studies, thus making at least the linkage disequilibrium statistic incomparable with the original studies and also out of the scope of this study. Third, the computing of the four latter statistics would have only brought along additional complexity to the computations and performance analysis from the viewpoint of this thesis. A detailed discussion on the choice of these statistics for this thesis is given in Section 3 and Section 5.

2.3.4 Output and file format compatibility

Concerning file format compatibility of bioinformatics in general, various bioinformatics software and their interplay is discussed in a recent review [Exc2006]. The opinions put forward in the article clearly point out the need for a common file format that would unify the shattered field of bioinformatics software and the respective output formats. As for `simuPOP`, the software already has inherent support for multiple file formats and powerful population genetics computer programs (Figure 3). A data exchange method for Arlequin software [Sta2001] would be quite straightforward to implement and would make a notable increase in the co-operation between the programs. Moreover, support for Arlequin would literally place `simuPOP` in the center of the interplay chart of a wide range of different bioinformatics software

[Exc2006].

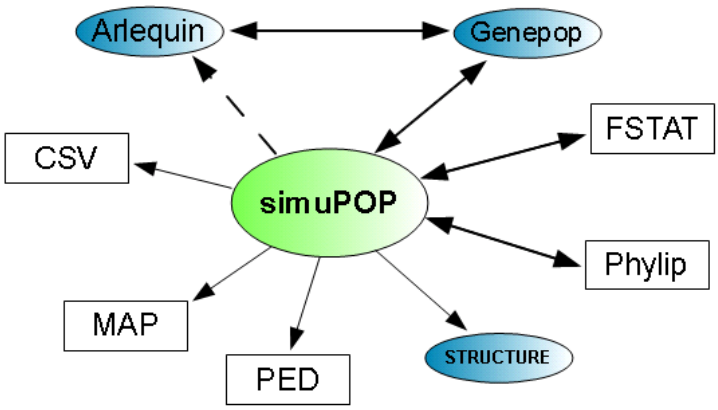

Figure 3: A chart showing simuPOP data exchange relations. The colored ovals represent software and the rectangles represent file formats. Export support for Arlequin is not inherently available in simuPOP, but the sample script referenced in this thesis does have a working export method.

2.4 Current capabilities and limitations of simuPOP

The simulation tool `simuPOP` [Pen2005] utilized in this study is likely to experience extensive use in future studies. Therefore, it is feasible to explore some of the limitations of `simuPOP` to detect boundaries of what can be simulated in the first place. In the development of `simuPOP`, most recent attention from the computational point of view has been paid in simulating long DNA sequences. The main issue often is the extensive use of computer random access memory (RAM) in computations that include millions of individuals with a detailed genome. `simuPOP` currently has support for multiallelic and biallelic simulation scenarios, but doesn't have direct support for binary mode optimizations on the scale presented in this thesis (see Section 3.4.2). Later versions of simuPOP (since release 1.0.8) offer a new method for simulating long DNA sequences but the method still is very limited in its approach.

`simuPOP` also is somewhat notorious for its steep learning curve in everyday use, where sufficient skill in Python programming is required. This and some other minor issues of `simuPOP` are pointed out in a recent Master's thesis [Heg2010] but not further discussed here.

When considering genetic simulation of large populations, the inbuilt memory allo-

cation of `simuPOP` becomes a problem, as the random access memory of the computer running the simulation quickly becomes overwhelmed by large population sizes and detailed genomes of individuals. The desirable future developments would be to reduce the memory load of storing individual genomes and allow greater population sizes in the simulation with individuals still having a detailed genome to simulate.

2.5 Methods for sequence data storage and retrieval

`simuPOP` is only one example of a software that by default stores everything it needs in random access memory in runtime. It is intuitive that for instance individual genomes may not be needed at hand all the time. Memory could be saved if the individual genomes could be stored outside random access memory or compressed temporarily and still quickly accessed when needed.

Storage and retrieval of a large collection of highly similar sequences has been studied in the past. University of Helsinki Department of Computer Science recently had a research group known as *SuDS* (Succinct Data Structures) for investigating these applications (http://www.cs.helsinki.fi/group/suds). In an example publication, a data structure known as the *suffix tree* is applied to storage of 36 repetitive sequences covering a total of 409 megabases of DNA sequence [Mak2010].

The payoff of suffix trees is that retrieval of information from the tree is fast: Desired information (a sequence) can be searched from the tree based on key values (suffixes) stored in the nodes. As areas of the search space can be identified based on the key values, parts of the tree where the search is guaranteed not to find the sequence can be excluded. The major overhead of suffix trees is the actual construction of the data structure from a collection of strings. An efficient, more accurately *linear-time*, algorithm for building suffix trees was developed in the University of Helsinki Department of Computer Science by Esko Ukkonen in the early 1990's [Ukk1992].

While suffix trees appear to have many advantages, the basic implementation of a suffix tree does not save any space. Actually, storing a sequence of length n consisting of characters A, C, G and T in a suffix tree takes $O(n \log n)$ bits of space [Val2007]. To improve the situation, compressing the sequcences stored in the suffix tree is one solution. A string processing method known as self-indexing [Mak2007] can achieve this, and the compression is promised to only have a *polylogarithmic slowdown* on the retrieval operations while still allowing for random access to the contents of the tree [Mak2010]. In practice, a method called *Burrows-Wheeler transform* [Bur1994]

is used to sort sequences in a collection into a format where they would be more amenable to compression. More accurately, similar elements (characters) in the sequences are placed closer to one another in the transformed order of the elements [Mak2007].

The overheads and limitations presented here do not concern only population genetic simulations, but almost every application where a collection of strings needs to be stored and compressed. Examples other than population simulations include genome alignment [Kur2004] and metabolic network reduction algorithms in pathway analysis [Zen2010].

2.6 A brief look into prehistory of Finland

This thesis uses a recently developed population simulation scenario concerning Finnish population history as an example case for investigating the throughput of `simuPOP` [Sun2010, Sun2013]. This was done in order to compare the results of different configurations of `simuPOP` against published summary statistics and to assure that the methods introduced in this thesis do not affect these statistics. For clarity, the geographical location of Finland is illustrated in Figure 4.

Figure 4: Reference map showing location of Finland (black) in northern Europe. Modern country borders as depicted in this map obviously have not existed before the historical times and have formed into their present state only very recently.

Finland has been inhabited for at least 10,000 years. The era 10,000 years ago is known in human prehistory as the *Stone Age* named after the most used material in the artifacts used by man at that time. The Stone Age is classically further divided into the *Mesolithic* (Early Stone Age, *circa* 11,000-6,000 years ago) and the *Neolithic* (Late Stone Age, *circa* 6,000-3,000 years ago) and is succeeded by the *Early Metal Period* (Bronze Age). The oldest radiocarbon dated evidence of human activity is from Joensuu in eastern Finland, a piece of a mammal bone which evidently has been burned in a fireplace [Pes2010]. The Ristola dwelling site in Lahti also is very old, though not radiocarbon dated. Another interesting site is Kristinestad, Finland: the so called *Susiluola* (literally translated as *wolf cave*) [Mag1998, Pur2004] in Karijoki. The wolf cave is the oldest human inhabitation site found in Northern Europe. There has been some debate as to whether human inhabitation existed in the wolf cave already before the Ice Age, more than 100,000 years ago, but this hypothesis lacks undisputed evidence at the time of writing.

Radiocarbon dated materials have provided information that indicate a population peak period in the Neolithic period, and before the Early Metal Period, the population appears to enter a decline [Tal2010]. The actual reasons for this decline are unknown. Deteriorating climate, wars, famine and an epidemic are all plausible reasons and are discussed in detail in the first of the cited papers of Finnish population history simulation [Sun2010] and references therein. A population bottleneck succeeding the decline also appears likely and may be one explanation for the lowered genetic diversity discovered in the Finnish population of modern day [Saj1996, Pal2009]. Reliable census population sizes concerning the Finnish population are available from 18th century onward and the total population first exceeded one million in the late 1800's [Ker2001]. Today the Finnish population totals to about 5,4 million individuals.

An isolate population consisting of northern and western Europeans has earlier been simulated as an isolate with `simuPOP` [PaK2011], although the individuals could not be directly associated with those originating from Finland. The actual population of Finland has also been simulated in in recent studies with `simuPOP`: Known census sizes were used in the simulation concerning the so called Savonian expansion in Finland applying also significant historical information such as small-scale migration habits of individuals in Finland of the past [Heg2010]. Next, the entire population history (*circa* 11,000 years) was simulated in [Sun2010] and further improved in [Sun2013]. In the latter three studies, the settings of population demography were based directly on the archaeological backround presented in this section. Conse-

quently, the scenario used in the simulations presented in this thesis is also based on the most recent publications [Sun2010, Sun2013] and is further discussed in Section 3.3.

3 Materials and methods

The main improvements to the workflow of `simuPOP` presented in this thesis are restricted to forward-in-time simulation of long sequences of DNA that consist from the four nucleotide bases (A, C, G and T). The improvements do not apply to other representations of the genome such as Y-chromosomal microsatellites or multi-allelic loci, though those were simulated in the original studies [Sun2010, Sun2013] and are also simulated here to replicate the memory loading as accurately as possible.

As this thesis investigates the performance of a software, allocation of external computer resources was needed. Testcases in a mere single tabletop or laptop computer environment might very well have been insufficient to test some of the real limitations of simulations with `simuPOP`. The Department of Computer Science of the University of Helsinki has recently acquired a high-throughput computer cluster now named as `ukko`. Login access to `ukko` is free for students and staff of the department, and the machine was extensively used in the testing, analysis and replication of the simulations described in this thesis.

The overview of the three methodological approaches to improve throughput of `simuPOP` are shown in Figure 5.

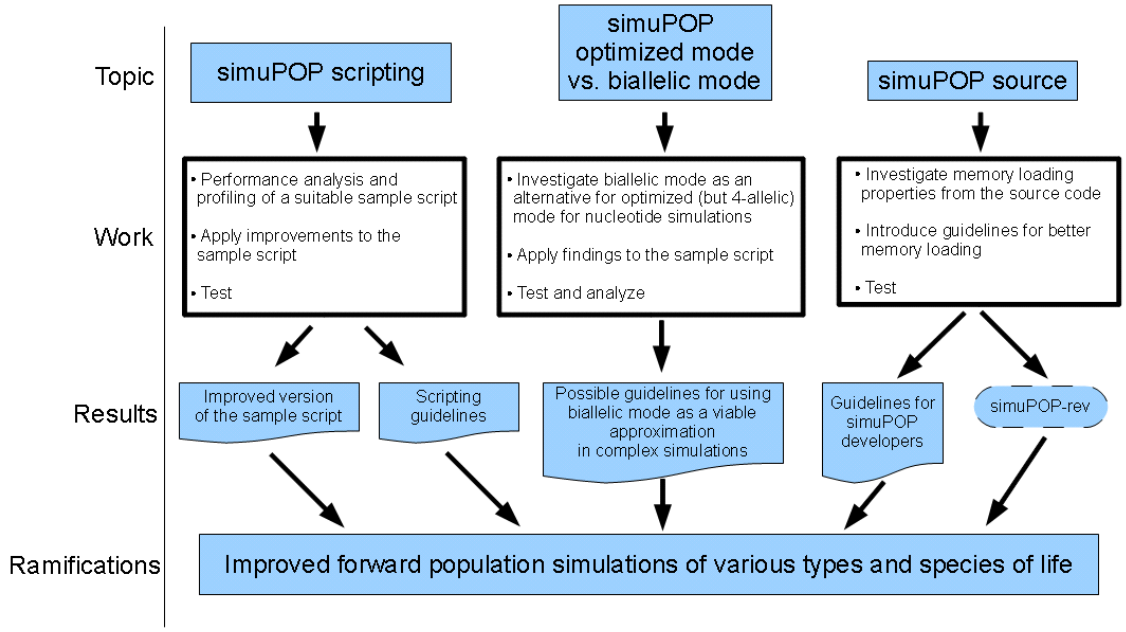

Figure 5: Overview of the three methods of this thesis to improve throughput of simuPOP.

3.1 Computing resources

`ukko` cluster of the Helsinki University Department of Computer Science was used extensively in testing the simulation scripts and software packages needed for profiling and performance testing. The system has a 64-bit GNU/Linux as operating system. The batch simulation runs and actual development of `simuPOP-rev` were also performed on the `ukko` cluster. Each login node of `ukko` has 16 Intel® Xeon® E5540 (2.53 GHz) processors available. As `simuPOP` has no support for parallelization, a single simulation run used exactly one of these processors. The core of `simuPOP` was checked for memory leaks using a software called Valgrind [Sew2005].

The latest release version of `simuPOP` (1.1.1) was downloaded and compiled from its `Subversion` repository in Sourceforge (http://www.sourceforge.net). In essence, this version was *forked* and the development versions of this *fork* are now designated as `simuPOP-rev`. The downloading instructions of `simuPOP-rev` are available online (http://simupop-rev.sourceforge.net).

3.2 Performance analysis

A built-in wall clock implementation of `simuPOP` as well as the central processing unit time (*user time*) console output command `time` were used to measure computing times. An openly available Python module called `guppy`, especially its `heapy` memory profiler [Nil2006], was used for the assessment of the memory loading. Moreover, the output of Unix command `top` was directed into a file in the final generation of each simulation run (details in Section 3.3.3). The command reports many real-time statistics such as CPU use and random access memory resident set sizes for processes. The computing times were set to measure the time elapsed to simulate from starting population to final population, or from generation 0 to generation 1,100 (details in Section 3.3.3). Time spent in calculating the statistics of interest after the actual population simulation were excluded from the performance analysis as the calculation here was largely user-defined and not a feature of `simuPOP` itself. The memory loading was set to measure the size of objects in memory when population size reaches 1 million (1,000,000) individuals.

In order to gain unbiased performance measurements, the actual wall clock simulation times (seconds) and memory loading (gigabytes) of the simulation were recorded from a set of 100 independent simulation replicates. The arithmetic averages and medians of the wall clock and user times were then computed and illustrated as boxplots where applicable.

3.3 Sample simuPOP script: Prehistory of Finland

The demographic outline used in the simulations of this thesis is based on the archaeological background introduced in Section 2.6. Many fixed events in the scenario such as population bottlenecks are based on genetic evidence [Saj1996] and evidence from radiocarbon dated archaeological artefacts [Tal2010], and are further supported by recent simulation studies that were done with `simuPOP` [Sun2010, Sun2013]. In the more recent of the studies [Sun2013], 24 different simulation scenarios were run. Regarding the scope of this thesis, a single simulation scenario labeled "E1" in the previous study was selected for simulations presented here. The characteristics of the E1 scenario include migration between subpopulations, external migration into the main population from neighbouring populations (details in Section 3.3.2) as well as a bottleneck size of 1,000 individuals at the narrowest point. It also was one of the scenarios that produced population genetic statistics closer to those observed

in authentic genetic data in the previous study [Sun2013]. An illustration of the general demographic model is given in Figure 6.

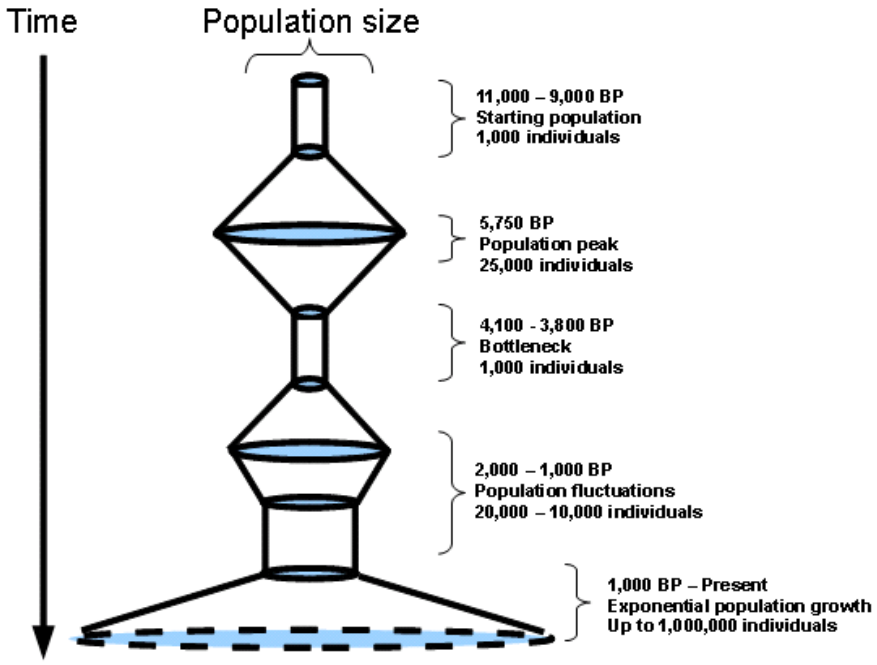

Figure 6: The general demographic model in the simulations performed in this thesis. The model corresponds to a specific scenario (E1) of a previous simulation study [Sun2013]. Subpopulation division is not visible in this illustration. *BP = years Before Present.*

The model simulation period begins at 11,000 years BP (=*Before Present*). The simulation period ends at 0 years BP, a time that roughly equals present day. The simulation proceeded in steps of 10 years. Thus, one whole simulation consisted of 1,100 steps in total. The simulation was repeated 100 times in all test runs, and statistics of interest (see Section 2.3.3) were then calculated after each simulation run. Concerning internal hierarchy of populations, Masatoshi Nei has pointed out that it is crucial to take population subdivision into account in population genetic analysis [Nei1973]. Thus, the main population was divided into several subpopulations as described in the following subsections.

3.3.1 Finnish population

The Finnish population was initialized with 1,000 individuals (500 males and 500 females) distributed evenly in two subpopulations: Saami and Other Finland (marked

as "Muu Suomi"). As it is currently impossible to accurately predict the actual genetic composition of the ancient individuals of the starting population, the initial values of the genomes were selected from a set of haplotypes listed in a Finnish population genetic study of actual genetic data [Hed2007], and corresponded with the previous academic estimates used in both previous simulation studies of the entire Finnish population prehistory [Sun2010, Sun2013].

3.3.2 Neighbouring populations

Migration of random individuals was added from three neighbouring populations of constant size: archaic European, archaic Scandinavian and Saami. The constant sizes for these populations were 50,000, 25,000 and 5,000 respectively. The compositions of the neighbouring populations were an exact match to those used in the previous simulation studies [Sun2010, Sun2013]. The neighbouring populations were simulated before the actual simulation run to act as static pools from which migrating individuals were then sampled. Unfortunately, it is out of reach for the current version of `simuPOP` to be able to simulate large background populations in runtime aside with the sizeable and growing Finnish main population.

3.3.3 Timeline of simulation

The simulation starts at 11,000 years before present time (BP). Two population bottlenecks were placed into the simulation, one at 4,100-3,800 BP and a second one at 1,500-1,300 BP. These bottlenecks are archaeologically justified and correspond with the simulation timeline of a previous study [Sun2010]. 1,000 years after the start of the simulation (on step 100), the Muu Suomi subpopulation is split into two: North-East subpopulation and South-East subpopulation. This enables subpopulation-specific migration from the neighbouring populations. The same subpopulation structure was also used in the most recent simulations of Finnish prehistory [Sun2013].

3.3.4 Simulation operators and parameters

The simulated populations were age-structured meaning that the generations overlapped. Female individuals had mating age from 20 to 40 years of age, and male individuals had mating age from 20 to 60. Moreover, the maximum age for an individual was 60 (6 simulation steps). In the actual simulation, the age information

was directly encoded into the individual information fields of individuals, one of the most powerful and flexible features of `simuPOP` (details discussed in Section 5.6).

Mating of individuals was set to be random, following the mating scheme used in previous simulations. The number of offspring for the two individuals chosen for mating is a random variable that follows a zero-truncated Poisson distribution with expected value of two (λ=2). A general mortality rate of 0.15 in every 10 years was introduced.

A mutation rate μ of 5.12×10^{-6} per site per simulation step was used for the simulated 631 bp (= *base pair*) mitochondrial DNA segment. This is the same as used in the previous studies [Sun2010, Sun2013], and is an arithmetic average of values based on literature on the subject [Hey2001, Sig2000]. Mitochondrial DNA was chosen as the simulated marker in the previous studies as a great deal of literature on the subject was available and, more importantly, because mitochondrial DNA is virtually free of recombination making it a more stable genetic marker. The mutation model used was that developed by Motoo Kimura [Kim1980], where transitions (nucleotide mutations A $\leftrightarrow$ G and C $\leftrightarrow$ T) are κ times as probable as transversions (other nucleotide mutations). Here, κ was fixed to value 0.5, apparently a standard default value for this mutator [Pen2005], and the same value as used in the previous studies mentioned above. The mutation matrix for the simulation is given in Table 1.

	A	**C**	**G**	**T**	**-**
A	-	$\mu/4$	$\kappa\mu/4$	$\mu/4$	0
C	$\mu/4$	-	$\mu/4$	$\kappa\mu/4$	0
G	$\kappa\mu/4$	$\mu/4$	-	$\mu/4$	0
T	$\mu/4$	$\kappa\mu/4$	$\mu/4$	-	0
-	0	0	0	0	1

Table 1: Mutation matrix applied in multi-allelic simulation modes. Here, μ=5.12×10^{-6} and κ=0.5 .

The mutation matrix in Table 1 apparently accounts for deletions in the sequences input to the mutator but deletions have zero probability of mutating further and no new deletions are introduced (see last column and last row in Table 1).

In order to maintain comparability of the computational load in the simulation, Y-chromosomal microsatellites incorporated in the original studies were also simu-

lated here. A respective mutation rate of 7×10^{-4} was used for the Y-chromosomal microsatellites [Kay2001] with a standard stepwise mutation model [Oht1973].

Migration between the subpopulations was also added with rates corresponding to internal migration of a patrilocal society. This means that migration probability is significantly higher for females than males. Gender-specific migration rates between subpopulations were deemed appropriate in a previous simulation study [Heg2010]. According to the estimate, it is 10 times more likely for female individuals than male individuals to migrate from one subpopulation to another. Moreover, it is assumed to be 10 times more likely for an individual to migrate to a neighboring subpopulation than to a subpopulation located farther away. The migration rates are presented in Table 2.

Migrate to	male	female
neighbouring	0.03	0.3
farther	0	0.03

Table 2: Migration rates for individuals into neighbouring and farther subpopulations during simulation.

The population genetic statistics calculated from the simulated data were *i)* population size, *ii)* number of haplotypes and *iii)* genetic diversity. Population size was fixed according to the general demographic model (see Figure 6 on Page 19) and was not further analyzed. The number of haplotypes and genetic diversity were calculated for the 631 bp of simulated mitochondrial DNA as the average of 10 random samples from the final (present day) population with a sample size of N=832. The sample size is an exact match to that used in a recent publication concerning the analysis of actual genetic data from Finnish subpopulations [Pal2009]. For the mitochondrion, the statistics reported in this thesis are the same as in that publication.

3.4 Methods for improving throughput of simuPOP

This thesis introduces three different ways to improve throughput of `simuPOP` (see Figure 5 on Page 16). These are discussed in detail in Sections 3.4.1, 3.4.2 and 3.4.3.

3.4.1 Scripting guidelines through performance analysis

Python programming language [Ros2011] is used as an interface language for operating `simuPOP`. A sample Python script for running the previously published simulations [Sun2013] was used. More accurately, the simulation scenario E1 of the previous publication was used in all simulations presented in this thesis. The scenario corresponds to that presented in Figure 6 in Section 3.3.3.

The simulation script was first inspected and critical parts of the code were evaluated. A performance analysis simulation set of 100 runs was then batched on the `ukko` cluster of Helsinki University's Department of Computer Science. Based on evaluating the script, the standard script was then modified by manually adding a memory management procedure known as *garbage collection* in each of the 1,100 simulation steps. In practice, this was done by importing the standard Python module for memory management:

```
import gc
```

The garbage collection was implemented by adding a simple Python subroutine that was called by the simulator in each generation just before the mating phase (see Figure 2 in Section 2.3.2) of the population:

```
def garbageCollect(pop):
  gc.collect()
  return True
```

Another 100 simulation runs were also batched with these settings. In addition, a further 100 simulation runs were batched on the `ukko` cluster using the biallelic (see Section 3.4.2) version of the simulation script. The modified versions of the script are available and can be obtained by contacting the author of this thesis.

3.4.2 Using simuPOP biallelic mode as an approximation

For more simple simulation scenarios, `simuPOP` has available a *biallelic* mode that utilizes binary vectors and bitwise operations instead of the 32-bit unsigned integer vectors used in default multi-allelic mode. In spite this mode is primarily designed to distinguish between segregating and preserved sites in a simulated genome, it may offer an alternative to approximate the multi-allelic mode while saving memory

on the same. In order to repeat the original simulation scenario introduced in 3.3, some major changes had to be introduced to the sample script. Most importantly, the representation of individual genomes was changed so that a single allele is represented by 2 one-bit fields (with value 1 or 0). The biallelic representations of the four nucleotide bases A, C, G and T are given in Table 3. Figure 7 illustrates how this configuration looks like inside the simulator.

Base	**Biallelic representation**
A	00
C	01
G	10
T	11

Table 3: Nucleotide base representations in the biallelic mode.

Actually, the exact coding of the nucleotide bases in the biallelic mode in Table 3 is irrelevant. This is due to the fact that the built-in mutator of `simuPOP` cannot identify the index (location within the genome) of the allele that is targeted for mutation [Pen2005]. Thus, should a mutation happen, point mutations of the two binary components of a single base in the biallelic mode are independent of one another by design. In a probabilistic sense, this produces the same amount of possible outcomes of a mutation event irrespective of which exact coding of the bases is used.

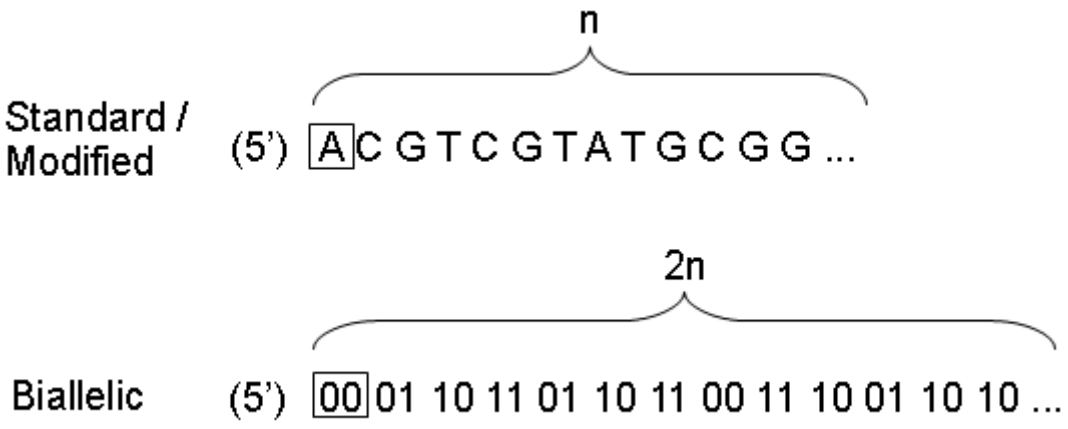

Figure 7: An illustration of the changes introduced into genomic representations of a single individual within the simulator when changing from the standard (4-allelic) mode to the biallelic mode. The rectangles outline the representation of a single nucleotide base.

Furthermore, the mutation model for the standard 4-allelic mode (see Table 1 in Section 3.3.4) obviously would not hold after the changes to the genome representation (Figure 7). This was resolved by introducing a modification where single nucleotide mutations of the standard 4-allelic mode are covered by a simplified mutation matrix. Without specifying which exact mutation happens, the probability that *some* mutation happens was interpreted as the sum of mutation probabilities of the standard 4-allelic mode (see any row of Table 1 in Section 3.3.4):

$$\mu/4 + \mu/4 + \kappa\mu/4 = (\mu + \mu + \kappa\mu)/4 = (\kappa + 2)\mu/4 \; . \tag{3}$$

Finally, the mutation rate for the biallelic mode was divided by two, as a single allele now is represented by two (binary) alleles (see Figure 7):

$$((\kappa + 2)\mu/4)/2 = (\kappa + 2)\mu/8 \; . \tag{4}$$

This sets the probability of *some* mutation taking place exactly the same as in the standard 4-allelic mode. The mutation matrix applied in running the biallelic mode is shown in Table 4.

	0	**1**
0	-	$(\kappa{+}2)\mu/8$
1	$(\kappa{+}2)\mu/8$	-

Table 4: Nucleotide base mutation matrix applied in the biallelic mode.

The `simuPOP` script written in Python for running the biallelic mode with these modifications is available and can be obtained by contacting the author of this thesis.

3.4.3 Changes in the source code of simuPOP

An independent development version of `simuPOP` now called `simuPOP-rev` was founded. The original aim of this was to direct the development of the simulation software towards resolving computer memory issues rather than solving population genetics issues. As the software license of `simuPOP` (General Public License) dictates, `simuPOP-rev` is also released under the very same license. The source code of `simuPOP-rev` is freely downloadable at http://simupop-rev.sourceforge.net (Figure 8).

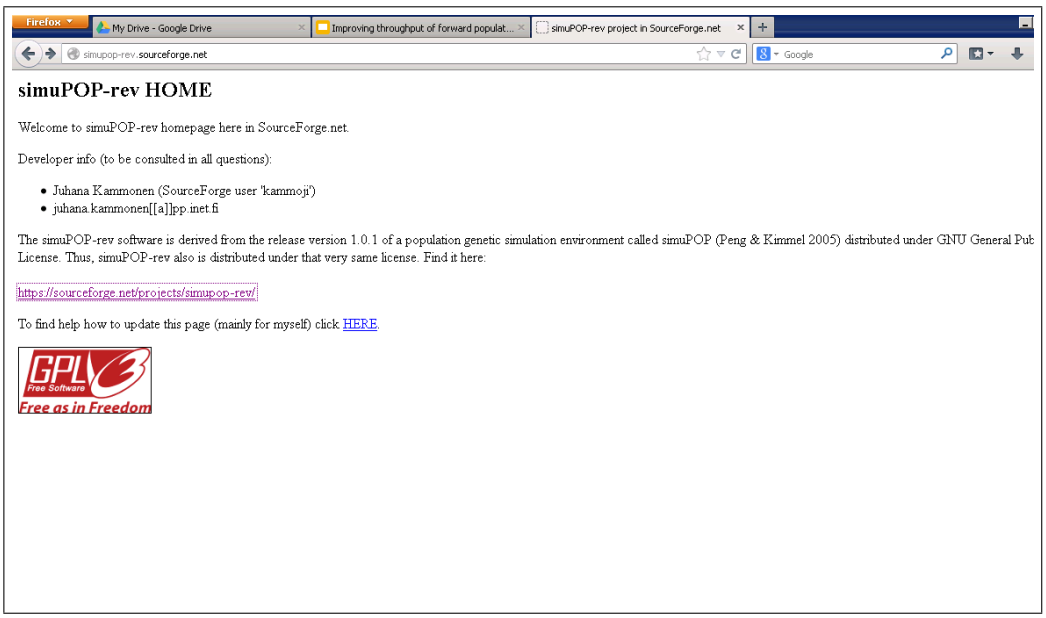

Figure 8: A screenshot of current simuPOP-rev index page online.

In previous simulation studies with `simupop` [Sun2010, Sun2013], the neighbour-

ing populations were implemented as static pools (see Section 3.3.2) from where migrators were randomly picked and added to the main population. Should the simulated population be considerably larger than 1,000,000 individuals, the simulation process would be indefinitely lengthened and, at least in an everyday tabletop or laptop environment, would quickly eat up all memory of the computer running the simulation.

At the time of writing this thesis, the development of `simuPOP-rev` is still in a very early stage. The future directions and ultimate goals of this development are discussed in Section 5.4.3.

4 Results

This section presents the results of the multiple methods applied in this thesis. A data visualization entity known as *boxplot* was chosen as the illustration method for many of the results in following subsections. The boxplots in this thesis follow the standard boxplot format: the box represents the limits of the first and the third quartile of the sample set values. The line inside the box depicts the median value of the sample set. The whiskers depict the limits of the 1,5 times interquartile range of the sample set at both ends. Small circles in the plot area are interpreted as *mild outliers* and stars as *severe outliers*.

While this section rather strictly pertains to reporting the results, an analysis of the results and general discussion follows in Section 5.

4.1 Performance analysis

The Valgrind [Sew2005] check of the `simuPOP` core did not indicate any potential memory leaks. The `heapy` memory profile browser screenshot of a single simulation run using the standard mode is shown in Figure 9 .

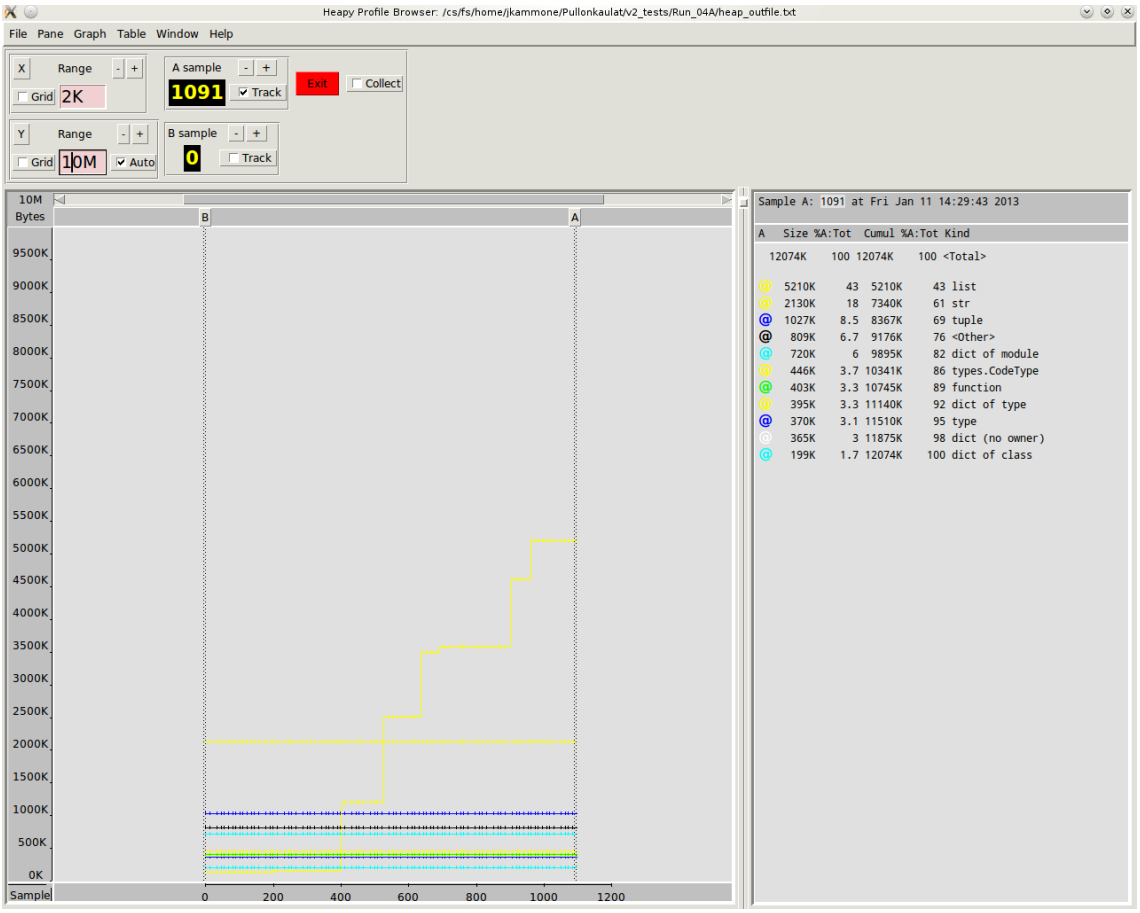

Figure 9: Screenshot of the heapy memory profile browser showing a single simulation run showing memory allocation over the 1,100 steps of simulation.

Using the built-in tools of `simuPOP` and exporting the outputs of system commands `top` and `time`, two statistics concerning performance of the software were recorded: computing time (both wall clock time and user time) and random access memory loading. The observed values for the computing times are given in Figure 10 and Figure 11.

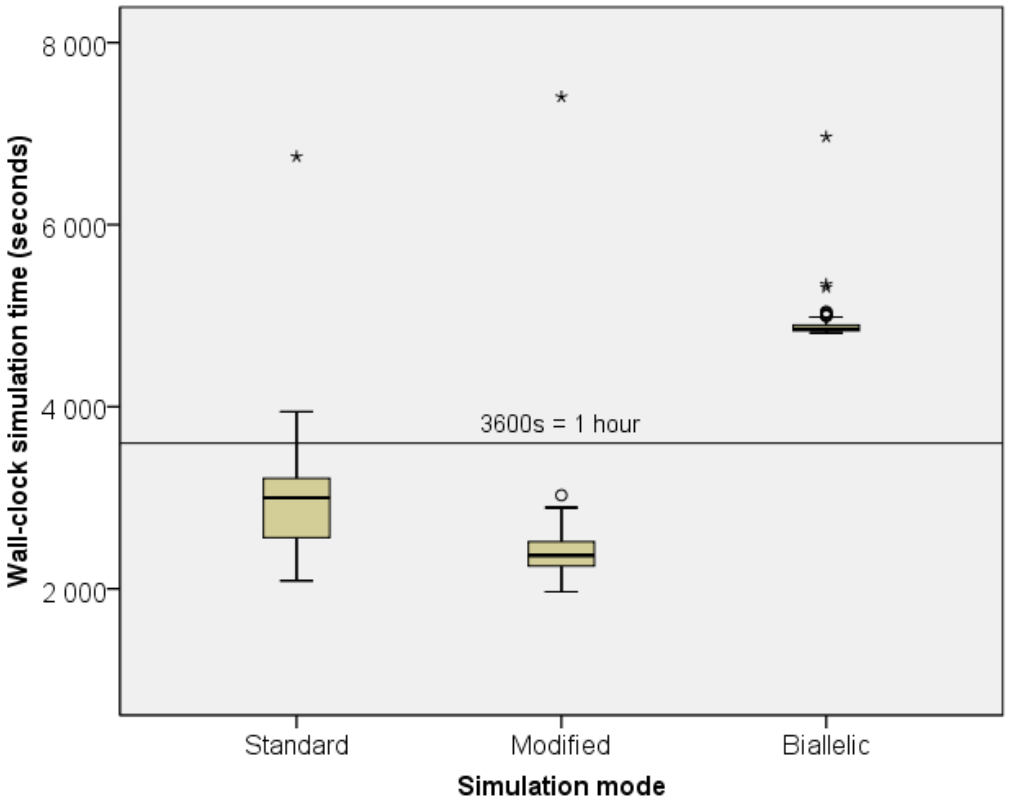

Figure 10: A boxplot of wall clock times (in seconds) observed in 100 runs of different simulation modes. The 3600 seconds (= 1 hour) reference line was added for visualization purposes.

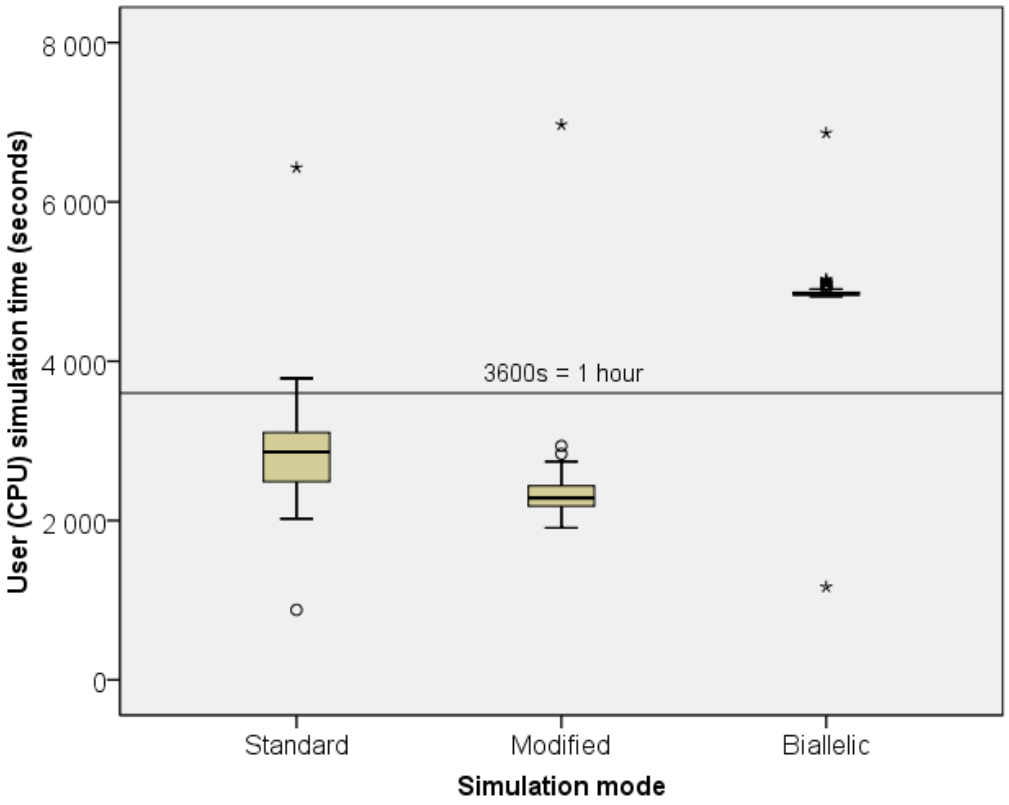

Figure 11: A boxplot of user times (in seconds) observed in 100 runs of different simulation modes. The 3600 CPU seconds (= 1 CPU hour) reference line was added for visualization purposes.

The observed values for memory loading are reported both in the output of `top` command as measured in final population (see Table 5) and in the maximum resident set size meaning the maximum memory load a single simulation process experiences during its lifetime (see Figure 12). Concerning memory loading, only average values are reported here for the `top` command output due to the extremely low variance of the values (Table 5).

Standard	Modified	Biallelic
2.90 *GB*	2.90 *GB*	0.95 *GB*

Table 5: Average random access memory resident set sizes for different simulation modes in final generation (population size *circa* 1,000,000 individuals). *GB = giga-bytes.*

Maximum resident set sizes during simulation runs are shown in Figure 12.

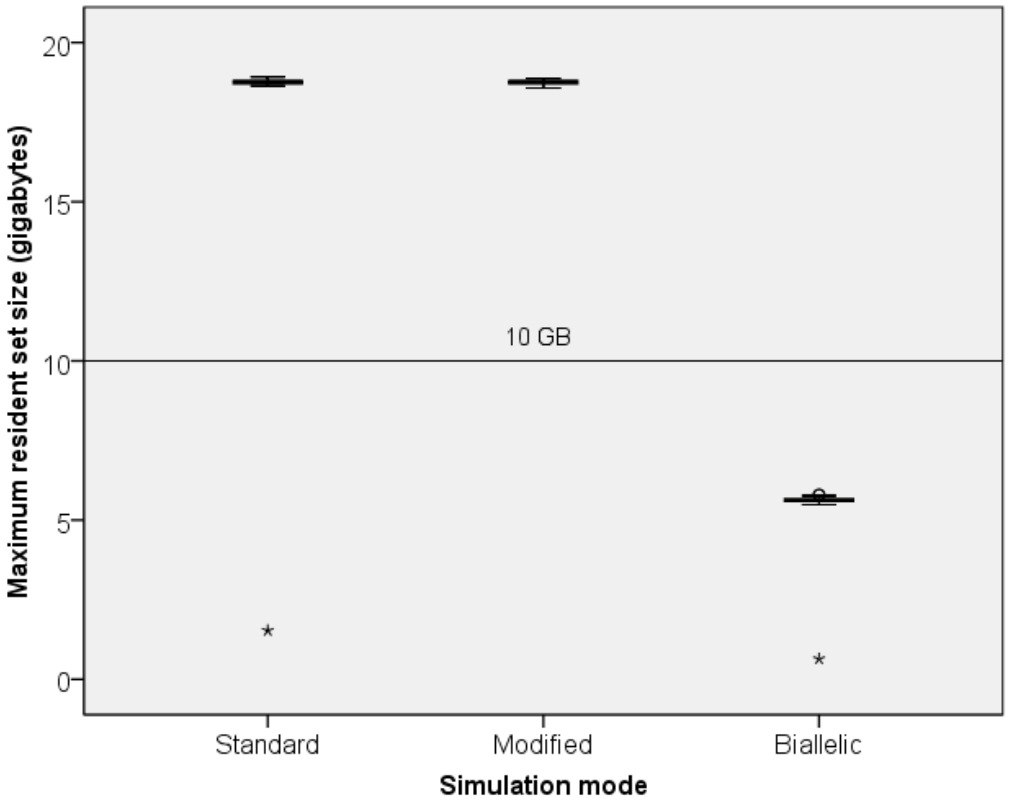

Figure 12: A boxplot of maximum random access memory resident set sizes observed in 100 runs of different simulation modes. The 10 GB reference line was added for visualization purposes (GB = *gigabytes*).

4.2 Using simuPOP biallelic mode as an approximation

Population genetic statistics introduced in Section 2.3.3 were calculated in the 100 performance analysis runs with both the standard mode and modified mode. A batch of 100 runs where the biallelic mode configuration (see Section 3.4.2) was also applied. Each of the simulation replicates were run on one node of the `ukko` cluster thus using three nodes in total.

4.2.1 Population genetic statistics

Population genetic statistics including the biallelic mode approximation described in Section 3.4.2 are shown below in Figure 13 and Figure 14.

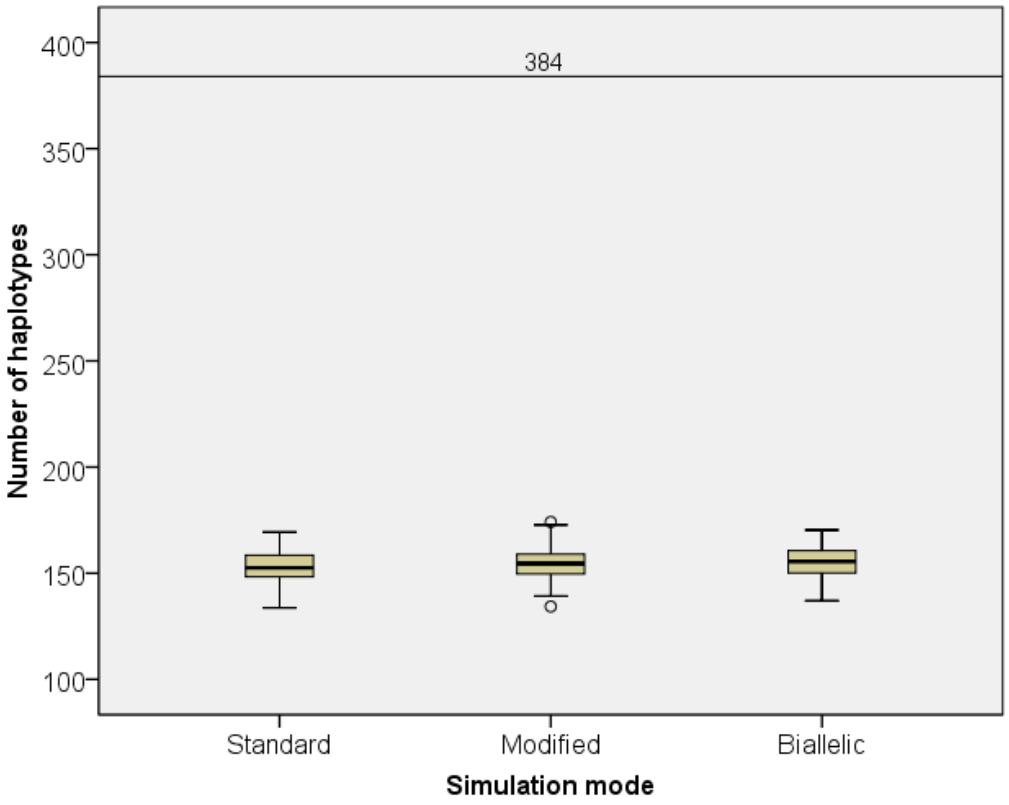

Figure 13: A boxplot of observed simulated number of mitochondrial haplotypes in final population in 100 simulation replicates. The aborted runs of standard and biallelic mode were omitted from the samples. Values of a single simulation run are computed as arithmetic averages of 10 independent samples (N=832). The horizontal line showing the value observed in genetic data [Pal2009] was added for reference.

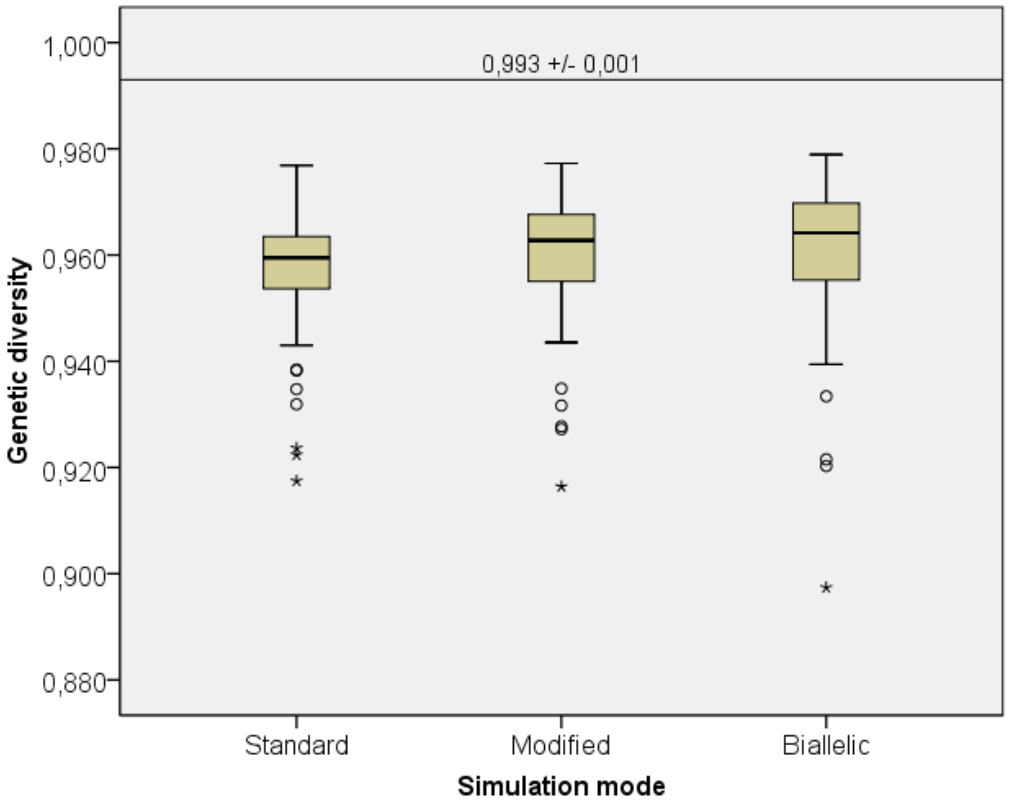

Figure 14: A boxplot of observed simulated mitochondrial genetic diversity in final population in 100 simulation replicates. The aborted runs of standard and biallelic mode were omitted from the samples. Values of a single simulation run are computed as arithmetic averages of 10 independent samples (N=832). The horizontal line showing the value observed in genetic data [Pal2009] was added for reference.

4.3 Changes in the source code of simuPOP

No actual major source code changes could be implemented within the scope of this thesis. However, an individual development version of `simuPOP` called `simuPOP-rev` was founded and is available for download at http://simupop-rev.sourceforge.net . `simuPOP-rev` will serve as a workbench for implementing the source code changes discussed in Section 5.4.3.

5 Discussion

Simulating populations of large and random quantities is computationally challenging. `simuPOP` will take the user to the very limits of computing in the sense that arbitrarily large simulations are possible, but with a sizeable enough population no computer can finish the work due to memory loading. A million individuals may well be enough to accurately simulate Finnish human population, but applications where an even larger population is simulated, such as human population of the whole Europe or a population of another species with inherently large population sizes (plants, bacteria), memory loading definitely becomes an issue.

Remembering the three methods of improving throughput of `simuPOP` presented in the previous sections (see Figure 15), this section discusses the selected methods and results and concludes with an outlook to future work. In addition, more elaborate ways to change the default random access memory storage of individuals are discussed in this section.

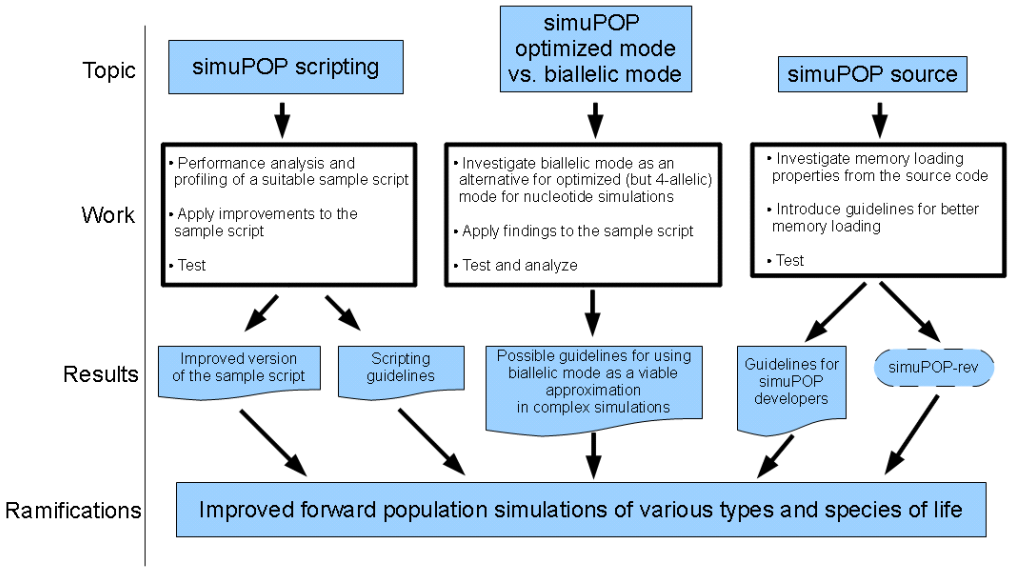

Figure 15: Overview of the three methods of this thesis to improve throughput of simuPOP.

5.1 Overall performance of simuPOP

The `simuPOP` population simulation environment stores individual genomes and other individual information into the random access memory (RAM) in runtime.

With a detailed and large genome in the simulation, the simulated population size may be limited into an infeasibly low number, especially when considering simulations of human populations. Evidently, care should be put on how to use `simuPOP` so that the memory available is used efficiently.

5.2 Simulation results

It is evident that the level of performance of `simuPOP` varies a lot depending on the mode in which the simulation is run (see Section 4.1). In general, the default multiallelic mode appears to be the fastest mode available as measured by both the wall clock time and user time, though reserving a threefold amount of random access memory as compared to the basic biallelic mode. However, swiftness of a single simulation run is greatly reduced when the simulation is run on the biallelic mode (see Figures 11 and 10 in Section 4.1). It would thus appear that the standard mode of `simuPOP` already is quite well optimized for speed.

By simply changing the `simuPOP` running mode from multi-allelic to biallelic without other changes introduced in this thesis would cause an approximation too rough to be compared with the standard mode. Evidently, all four nucleotide bases A, C, G and T are present in vast quantities in hypervariable regions of the mitochondrion. This diversity cannot be captured with only a single bit representing an allele. This was also apparent from the resulting population genetic statistics of a small batch of crude test runs not documented in this thesis.

However, applying the biallelic mode configured as described in Section 3.4.2 results very similar population genetic statistics as compared to the standard and modified modes (see Figures 13 and 14 in Section 4.2.1). The strength of this approach definitely is that although major changes to the actual sample simulation script were needed and an approximation of the mutation model was required, there was no need to modify the actual source code level mechanism that implements the nucleotide mutations. Thus, efficient population simulations can be implemented with the current release version of `simuPOP`.

The most important payoff of running the biallelic mode approximation is the reduced use of random access memory during a single simulation run. The biallelic mode consistently uses three times less random access memory during a single simulation run, as evidenced by Table 5 and Figure 12 in Section 4.1.

Finally, a notable amount of outliers were identified in the computing times in all

three of the simulation modes (see Figures 10 and 11 in Section 4.1). A single simulation run apparently aborted in both standard and biallelic modes resulting in the two lower-than-usual instances of user times (Figure 11) in the respective modes. The outliers are not visible in the wall clock times (Figure 10) as the runs aborted well before reaching the final generation where the wall clock was set to stop. Thus, the times were never recorded. Moreover, all three simulation modes had one run out of 100 replicates that took almost double the time than rest of the computing times sample set (see Figures 10 and 11 in Section 4.1). Where population genetic statistics were produced, the results of the outlier runs were included in the population genetic analysis. Obviously, the two aborted runs did not produce these statistics and were omitted from the calculation of the statistics (see Figures 13 and 14 in Section 4.2.1). As expected, the aborted runs also were visible in the memory loading measurement results, where two simulation instances (one in standard and one in biallelic mode) showed a lower-than-normal memory loading (see Figure 12 in Section 4.1).

While the methods used in this thesis (see Section 3) were unable to diagnose the exact source of the outlier simulation times or the reasons why some runs aborted, the author of this thesis is aware of a certain level of random behaviour when using `simuPOP` in supercomputer environment: A small amount of aborted runs was constantly observed while running the simulations reported in the original studies [Sun2010, Sun2013]. In the studies this was compensated by performing additional simulation runs.

5.3 Validity of the simulation model

Validity of a computer simulation model depends on many things, beginning from software and hardware used to the most minuscule factors that might affect the outcome of the simulation. Admittedly, no computer simulation model should be developed without implementing verification and validation (V & V) on the model. V & V lays basis on the development, iteration and later also backtracking of the model. Sargent [Sar2005] has suggested a collection of approaches to systematically perform V & V. At this point of the development the model that implements the scripting guidelines and approximations introduced in this thesis should be considered as the first version or *zero version*. Where applicable, the following development versions will then be predisposed to meticulous V & V.

Another topic to discuss is the validity of the demographic scenario presented in

investigating population history of Finland (see section 3.3). Whether the current genetic composition of Finns is the result of population demographic events such as bottlenecks, founder effects, prolonged isolation and/or genetic drift may be under debate. Nevertheless, the archaeological justification for significant population fluctuations during the early history of Finns cannot be disputed, as discussed in [Sun2008].

The population genetic statistics from the simulations performed in this thesis (see Figures 13 and 14 in Section 4.2.1) are quite far away from those observed in actual genetic data [Pal2009]. This deviation also was observed for the simulated portion of mitochondrial DNA in the original studies [Sun2010, Sun2013]. However, only a single population demographic scenario was studied in this thesis. Moreover, the original studies show that certain simulation scenarios are indeed closer to the observed values than others. A good interpretation of the results distinguishes between those sets of simulation parameters that yield the deviating outcomes. In case some scenario appears closer to real world than others, one should already have a good idea of why it is closer. Consequently, even if some simulation model produced exactly the observed genetic profile, simulation results never should be considered as absolute proof that the scenario is completely correct. The admittedly basic population genetic statistics reported in this thesis (see Section 3.3.4) also capture only a portion of the diversity in the population under investigation.

It is the author of this thesis' opinion that forward population simulations are a very feasible way of simulating Finnish population history. So far, the approach using `simuPOP` and previous work with it cited extensively in this thesis are the most well documented ones. Still, other approaches, for instance the serial coalescent introduced in section 2.3.1, to this kind of simulation are encouraged.

5.4 Improving throughput of simuPOP

`simuPOP` [Pen2005] has open source code and the software is under constant development. New versions of the software are released in intervals of approximately five months. This has given this author a great opportunity to contribute to the development of the software. This author has been in regular correspondence with the actual developers and the user community and many new features have been added to the software based on the experiences and needs expressed.

It is evident from the performance analysis that simulating a considerable amount

of DNA sequences of even 600 base pairs in length with `simuPOP` currently requires an intolerably large amount of computer memory (see Figure 12 and Table 5 in Section 4.1). Evidently, the extensive use of memory is caused by `simuPOP` storing the individual genomes and other information in random access memory. No actual memory leaks were found in the Valgrind [Sew2005] check of `simuPOP` core. However, Valgrind does not reveal data structure deficiencies. The memory profile produced by the `heapy` memory profiler (see Figure 9) was also inconclusive as to which lower level elements require the most memory (see Section 5.4.3).

5.4.1 Scripting guidelines through performance analysis

It was evident from the performance analysis that regular manual garbage collection has no meaningful effect on the memory use of `simuPOP` (see Table 5 and Figure 12 in Section 4.1). According to Figures 10 and 11 in Section 4.1 however, the garbage collection reduces the variance of both wall clock time and user time. Also the overall distribution and median value of elapsed simulation times appear notably lower as compared to the standard mode (Figures 10 and 11) in the 100 simulation replicates. Thus, introducing manual garbage collection in the simulation run at each generation just before the mating phase of the generation does have an effect on simulation times and may at best speed up overall running times, especially with a very large amount of simulation replicates. It should be considered remarkable that this was achieved with only very small changes into the actual simulation script (see Section 3.4.1).

5.4.2 Using simuPOP biallelic mode as an approximation

Through the suggested changes into the mutation model (see section 3.4.2), the biallelic mode approximation of the genome actually produces very similar population genetic statistics as compared to the standard multiallelic mode (see Figures 13 and 14 in Section 4.2.1). Thus, at least for simulating a sequence of DNA represented by the four bases A, T, C and G, the biallelic mode approximation is a viable alternative with a notable reduction in memory use. Moreover, the inbuilt method in `simuPOP` that implements the actual point mutations (see Section 3.3.4) does not need to be changed at all for the approximation to work. A setback of the biallelic mode is that the computing times of a single simulation run are almost doubled as compared to the other simulation modes (see Figures 10 and 11 in Section 4.1). Nevertheless, the

apparently linear reduction in memory use clearly is a notable payoff, especially in large-scale simulations of massively large populations.

5.4.3 Changes in the source code of simuPOP

The current release version of `simuPOP` supports allele coding for 1-bit, 8-bit and 32-bit fields [Pen2005]. Thus, there is no direct support for the biallelic mode as described in this thesis, although the mode could be implemented here by modifying the simulation script with the methods presented in Section 3.4.2 and illustrated in Figure 7. Evidently, the biallelic configuration results in a consistently threefold reduction in memory use at least for the simulation scenario presented in this thesis (see Section 4.1). Consequently, source code level changes in `simuPOP` to support the biallelic configuration would be highly recommended.

The `heapy` memory profile of the simulation run clearly shows that Python data structures known as `list` reserve more memory than any other data structure (see Figure 9 in Section 4.1). While the rather low memory loading values observed for the data structure `list` are not of great concern in this simulation instance, this behaviour should be noted and controlled when designing simulation scenarios with large population sizes. Moreover, `heapy` only shows allocated Python objects in the profile and not the lower level C/C++ [Str2000] objects. This is why the total memory use in Figure 9 is much smaller as compared to the memory usage measured in the performance analysis (see Figure 12 and Table 5 in Section 4.1).

Memory allocation system of `simuPOP` could also be changed to use general-purpose and region-based allocators called *reaps* [Ber2002], that in some instances are more efficient than even custom implementations of memory allocation.

Another tempting approach would be to see whether storing entire populations into a database during simulation and retrieving individual information from there when needed would improve throughput of `simuPOP`. Python does have built-in support for databases via a method known as *shelving* [Ros2011]. Database storage of genomes and individual information would solve the random access memory issues of massive population simulations. Main concern of this approach is that despite reduced memory use, the actual simulation run may become overly slow.

Finally, a new data structure for storing genomes may be improve memory-efficiency. This is another possible direction to consider for future source code changes. A compressed suffix tree based approach [Mak2010] outlined in Section 2.5 may fasten

retrieval of individual information and speed up a population genetic simulation. This would especially suit population genetic simulation scenarios where the diversity within the collection of the simulated sequences would be very low, and mutations and recombination could be considered as rare events.

In a hypothetical example of the compressed suffix tree approach, consider a collection of million sequences of text, 500,000 of which are labeled `sequence1` and are identical to one another, and the residual 500,000 sequences are labeled `sequence2` and also are identical to one another. A representation of the collection could look like this:

```
{sequence1, sequence1, sequence1, sequence1, sequence1, ...
 sequence2, sequence2, sequence2, sequence2, sequence2, ...}
```

with both of the sequences repeated 500,000 times. In contrast, the compressed suffix tree approach would eventually represent (and replace) the whole sequence collection as:

```
{500,000 x sequence1, 500,000 x sequence2}
```

This is a format that contains all the essential information of the collection. Although the presented example is an extreme one, an application where data is represented even in a remotely similar manner than this is likely to experience major improvements in space (in essence, memory) consumption with the compressed suffix tree approach.

The downside of the approach is that in practice, the memory use may actually increase, although the allocated memory would then be used to perform information retrieval operations more efficiently [Val2007]. The *polylogarithmic slowdown* feature [Mak2010] of the compressed suffix tree approach further suggests that compressed data structures may be a promising general direction of source code level development.

The author of this thesis has been in regular correspondence with the developers of `simuPOP`. They have unofficially suggested that instead of storing individual genomes

the users could be directed to storing only locations of mutations in the genome instead of the whole genome. Indeed, `simuPOP` release version 1.1.1 contains a module that can implement this. However, applying the module would require more elaborate changes to the genomes of individuals than what is presented in this thesis. Furthermore, simulating genome segments that have a high mutation rate such as the hypervariable regions of the mitochondrion would hardly benefit from this approach.

5.5 Simulating Finnish population with neighbouring populations

The short-term goal of simulations concerning the population history of Finland would be to simulate the neighbouring populations in runtime instead of static pools of migration (see Section 3.3.2) and to see how that changes the memory loading and results. So far, the population size limit due to individual genomes stored directly in random access memory has prevented this. The biallelic mode approximation intoduced in Section 3.4.2 and discussed earlier in this section may be the first step towards this kind of simulation, at least for the timespan where the population sizes remain relatively small. Still, `simuPOP-rev` will declare the goal of making real-time simulation of neighbouring populations possible also with population sizes exceeding 1,000,000 individuals.

5.6 Advanced use of individual information fields

The support of `simuPOP` for individual information other than genomic representations is one of the most powerful features of the `simuPOP` software. The previous simulations [Sun2010, Sun2013] as well as the simulations run in this thesis make an example of this by encoding age of individuals into the information fields (for details, see Section 3.3.4). Potentially any information concerning the simulated individuals can be encoded into these fields in the form of basic data types: integers, floating point numbers and characters. The information can then be processed and manipulated over the course of the simulation limited only by the interface of the scripting language Python. Python can be considered as a very expressive, easy-to-learn and easy-to-read programming language [Ros2011]. Many simulation software other than `simuPOP` would require source code level changes to be able to incorporate information in the same manner [Pen2005]. It is here where the flexibility of

simuPOP becomes apparent. simuPOP definitely seems to be a more general piece of population simulation software with potential applications even beyond population genetics. simuPOP-rev also is bound to greatly benefit from this feature and will make extensive use of it during the development.

5.7 Conclusions and future work

This thesis presented a review of literature and basic methods concerning *in silico* population genetic simulations, especially those of human populations. Multiple methods of improving one of the forward-in-time population simulators, simuPOP, were suggested and some of them were even implemented in practice and tested. As some of the improvements were deemed to require source code level changes, an independent development version of the simulator called simuPOP-rev was founded as a workbench for these changes.

Evidently, simuPOP is a suitable piece of software for very complex population simulations. Parallelizing the workflow of simuPOP, despite introducing a whole new field of problems on the same, would bring the power of supercomputing within the grasp of the simulation efforts presented. Especially the biallelic mode would greatly benefit from parallelization as more time-consuming simulation runs could then be run in parallel in multiple nodes of a computer cluster. simuPOP-rev will make an attempt towards this kind of development. The ukko cluster of the Department of Computer Science has proven itself to be a suitable testbed for testing the simulator.

Meanwhile, the development of simuPOP-rev will go on, and the author of this thesis has been recently inspecting the possibility to plug in a Python module for integrating the enhanced simuPOP-rev with a piece of animation software for visualization purposes. This addition would also bring the user interface of the simulator closer to potential users. Individual information fields are used extensively in this part of the development.

Answering the need for a more general file format not only in population genetic simulations, but also in the whole field of bioinformatics is another project that a prominent software developer could undertake. Evidently, some converter applications already exist [Exc2006], but must be kept up to date meticulously to keep pace with the growing number of bioinformatics software.

Improvements introduced in the future by simuPOP-rev enable more extensive simulation of populations without compromising the well-documented user interface of

the original version of `simuPOP`. Being a simulation framework that inherently enables incorporation of data from very different disciplines via individual information fields, the enhanced `simuPOP-rev` also encourages multidisciplinary simulation enterprises in the scientific community. Moreover, the lines of development presented in this thesis are likely to ever improve troughput of population genetic simulations of various species in life.

Acknowledgements

Writing this thesis was the hitherto greatest undertaking in my life. While it took me only about 6 months to actually write this thesis, the painstaking grounding work lasted several years. While feeling extremely discouraged and alone at times, I was not to give up hope that relentless effort would pay off some day. This thesis stands as the product of that effort. I want to thank my closest colleague Tarja Sundell and a good friend of mine Martin Heger who introduced me to the world of population simulations and who both have greatly inspired and supported this work. The experience gained will prove to be of great value for me in the future. I deeply hope that among many other things, the future will see me as a skilled general simulation software developer. Indeed, I have a feeling that the improvements presented in this thesis may also be easily applicable to software other than `simuPOP`.

I want to thank my thesis supervisor Mikko Koivisto for most valuable insights and instructions concerning the workflow, methods, visualizations and language of this thesis. I want to thank DSc Olli-Pekka Smolander for thoroughly inspecting the text in spite of a tight final schedule and a very short notice. I would also like to thank staff of DNA Sequencing and Genomics laboratory of the Insititute of Biotechnology - especially my superiors Petri Auvinen and Lars Paulin - for allowing me sufficient time to gracefully succeed in the balancing act of finishing this thesis and completing my other tasks at the lab. In addition, the experimental part of this thesis concerning `simuPOP-rev` was inspired by Linus Torvalds and his words of "how real men only program operating system kernels" as he visited my department in the October of 2012. Finally, I want to acknowledge the remote summer cottage of our family located in southern Savonia, Finland, which during the writing became a personal asylum and a birthplace for many of the great ideas and solutions to problems presented in this thesis. Several other people close to me have also helped me to reach a positive outcome in this long project. Thank you all!

The original simulation script, modified script and script for the biallelic approximation mode written in Python are available upon request by contacting the author of this thesis. `simuPOP-rev` is downloadable at http://simupop-rev.sourceforge.net

References

Bal2001 Balloux, F., EASYPOP (version 1.7): A computer program for population genetics simulations. *Journal of Heredity*, 92,3(2001), pages 301–302+.

Bur1994 Burrows, M. and Wheeler, D., A Block-Sorting Lossless Data Compression Algorithm. Technical Report, Systems Research Center, Palo Alto, California, 05 1994.

Bea2002 Beaumont, M. A., Zhang, W. and Balding, D. J., Approximate Bayesian computation in population genetics. *Genetics*, 162,12(2002), pages 2025–2035.

Ber2002 Berger, E. D., Zorn, B. G., McKinley and S., K., Reconsidering custom memory allocation. *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, New York, NY, USA, 2002, ACM, pages 1–12.

Cha2009 Charlesworth B., Effective population size and patterns of molecular evolution and variation. *Nature Reviews Genetics*, 10,3(2009), pages 195–205.

Cri1970 Crick, F. H. C., Central dogma of molecular biology. *Nature*, 227,5258(1970), pages 561–563.

Doi2004 Doit, C., Loukil, C., Simon, A.-M., Ferroni, A., Fontan, J.-E., Bonacorsi, S., Bidet, P., Jarlier, V., Aujard, Y., Beaufils, F. and Bingen, E., Outbreak of Burkholderia cepacia bacteremia in a pediatric hospital due to contamination of lipid emulsion stoppers. *Journal of clinical microbiology*, 42,5(2004), pages 2227–2230.

Exc2000 Excoffier, L., Novembre, J. and Schneider, S., Simcoal: a general coalescent program for simulation of molecular data in interconnected populations with arbitrary demography. *Journal of Heredity*, ,91, pages 506–509.

Exc2006 Excoffier, L. and Heckel, G., Computer programs for population genetics data analysis: a survival guide. *Nature Reviews Genetics*, 7,7(2006), pages 745–758.

Gra1996 Gravel-Tropper, D., Sample, M., Oxley, C., Toye, B., Woods, D. E. and Garber, G. E., Three-year outbreak of pseudobacteremia with Burkholderia cepacia traced to a contaminated blood gas analyzer. *Infection control and hospital epidemiology*, pages 737–740.

Hed2007 Hedman, M., Brandstätter, A., Pimenoff, V., Sistonen, P., Palo, J., Parson, W. and Sajantila, A., Finnish mitochondrial DNA HVS-I and HVS-II population data. *Forensic Science International*, 172,2-3(2007), pages 171–178.

Heg2010 Heger, M., Population Genetic Simulations of the Savonian Expansion in Finland. Master's thesis, University of Helsinki, 2010.

Hol2009 Holsinger, K. and Weir, B., Genetics in geographically structured populations: defining, estimating and interpreting FST. *Nature Reviews Genetics*, 10,9(2009), pages 639–650. URL `http://dx.doi.org/10.1038/nrg2611`.

Hey2001 Heyer, E., Zietkiewicz, E., Rochowski, A., Yotova, V., Puymirat, J. and Labuda, D., Phylogenetic and familial estimates of mitochondrial substitution rates: study of control region mutations in deep-rooting pedigrees. *American Journal of Human Genetics*, 69,5(2001), pages 1113–1126.

Job2013 Jobling, M., Hollox, E., Hurles, M., Tyler-Smith, C. and Kivisild, T., *Human Evolutionary Genetics*. Taylor & Francis Limited, 2013. URL `http://books.google.co.uk/books?id=UvUNPAAACAAJ`.

Ker2001 Kere, J., Human population genetics: lessons from Finland. *Annual Review of Genomics and Human Genetics*, 2,2(2001), pages 103–28.

Kim1980 Kimura, M., A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16,1(1980), pages 111–120.

Kur2004 Kurtz, S., Phillippy, A., Delcher, A., Smoot, M., Shumway, M., Antonescu, C. and Salzberg, S., Versatile and open software for comparing large genomes. *Genome biology*, 5,2(2004), pages R12+.

Kay2001 Kayser, M. and Sajantila, A., Mutations at Y-STR loci: implications for paternity testing and forensic analysis. *Forensic Science International*, 118,2-3(2001), pages 116–121.

Mag1998 Magnusson, E., Människor i Norden redan före istiden. *Forskning & Framsteg*, 1,1(1998), pages 4–5.

Mak2007 Mäkinen, V. and Navarro, G., Implicit compression boosting with applications to self-indexing. *In Proc. SPIRE'07, LNCS 4726*, 2007, pages 229–241.

Mak2010 Mäkinen, V., Navarro, G., Sirén, J. and Välimäki, N., Storage and Retrieval of Highly Repetitive Sequence Collections. *Journal of Computational Biology*, 17,3(2010), pages 281–308.

Nei1973 Nei, M., Analysis of gene diversity in subdivided populations. *Proceedings of the National Academy of Science USA*, 70,70(1973), pages 3321–3323.

Nil2006 Nilsson, S., Heapy: A Memory Profiler and Debugger for Python. Master's thesis, Linköping University, Department of Computer and Information Science, 2006.

Nie2001 Nielsen, R. and Wakeley, J., Distinguishing migration from isolation: a markov chain monte carlo approach. *Genetics*, 158,1, pages 885–896.

Oht1973 Ohta, T. and Kimura, M., A model of mutation appropriate to estimate the number of electrophoretically detectable alleles in a finite population. *Genetical Research*, 22,10(1973), pages 201–204.

Pen2007 Peng, B., Amos, C. I. and Kimmel, M., Forward-time simulations of human populations with complex diseases. *PLoS Genetics*, 3,3(2007), page e47.

PaK2011 Palin, K., Campbell, H., Wright, A., Wilson, J. and Durbin, R., Identity-by-descent-based phasing and imputation in founder populations using graphical models. *Genetic epidemiology*, 35,8(2011), pages 853–860.

Pen2005 Peng, Bo and Kimmel, M., simuPOP: a forward-time population genetics simulation environment. *Bioinformatics*, 21,18(2005), pages 3686–7.

Pes2010 Pesonen, P., Joensuu (Eno) Rahakangas 1, Kivikautisen asuinpaikan kaivaus. Technical Report, National Board of Antiquities, archive of archaeology, Finland, 05 2010.

Pal2009 Palo, J., Ulmanen, I., Lukka, M., Ellonen, P. and Sajantila, A., Genetic markers and population history: Finland revisited. *European Journal of Human Genetics*, 17,10(2009), pages 1336–46.

Pur2004 Purhonen, P., Lyhyesti Kristiinankaupungin Susiluolasta. *Tieteessä tapahtuu*, 8,8(2004), pages 48–50.

Ray2009 Ray, N. and Excoffier, L., Inferring past demography using spatially explicit population genetic models. *Human Biology*, 81,2-3(2009), pages 141–157.

Sar2005 Sargent, R. G., Verification and validation of simulation models. *WSC '05: Proceedings of the 37th conference on Winter simulation*. Winter Simulation Conference, 2005, pages 130–143.

Sew2005 Seward, J. and Nethercote, S., Using Valgrind to detect undefined value errors with bit-precision. *Proceedings of the USENIX'05 Annual Technical Conference*, 2005.

Sha2009 Shapiro, J., Revisiting the central dogma in the 21st century. *Annals of the New York Academy of Sciences*, 1179,1(2009), pages 6–28.

Sig2000 Sigurdardottir, S., Helgason, A., Gulcher, J. R., Stefansson, K. and Donnelly, P., The mutation rate in the human mtDNA control region. *American Journal of Human Genetics*, 66,5(2000), pages 1599–1609.

Sun2010 Sundell, T., Heger, M., Kammonen, J. and Onkamo, P., Modelling a Neolithic population bottleneck in Finland: A genetic simulation. *Fennoscandia Archaeologica*, 25,1(2010), pages 3–19.

Sun2013 Sundell, T., Kammonen, J., Heger, M., Palo, J. and Onkamo, P., Retracing Prehistoric Population Events in Finland Using Simulation. *CAA 2012 : Proceedings of the 40th International Conference in Computer Applications and Quantitative Methods in Archaeology*, 2013, pages 93–104.

Saj1996 Sajantila, A., Salem, A. H., Savolainen, P., Bauer, K., Gierig, C. and Pääbo, S., Paternal and maternal dna lineages reveal a bottleneck in the founding of the Finnish population. *Proceedings of the National Academy of Science USA*, 93,21(1996), pages 12035–9. URL `http://www.biomedsearch.com/nih/Paternal-maternal-DNA-lineages-reveal/8876258.html`.

Sta2001 Stankovski, Z., Arlequin: an integrated Java application. *JGI '01 Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, 2001.

Str2000 Stroustrup, B., *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, third edition, 2000.

Sun2008 Sundell, T., Populaatiopullonkaulat kivikauden Suomessa - arkeologinen ja geneettinen tulkinta. Master's thesis, University of Helsinki, 2008.

Tal2010 Tallavaara, M., Pesonen, P. and Oinonen, M., Prehistoric population history in eastern Fennoscandia. *Journal of Archaeological Science*, 37,2(2010), pages 251 – 260. URL `http://www.sciencedirect.com/science/article/pii/S0305440309003409`.

Tui2005 Tuimala, J., *Bioinformatiikan perusteet*. CSC - Finnish IT-center for Science, 2005. ISBN 952-5520-07-2.

Ukk1992 Ukkonen, E., Constructing suffix trees on-line in linear time. *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1 - Volume I*, Amsterdam, The Netherlands, 1992, North-Holland Publishing Co., pages 484–492.

Val2007 Välimäki, N., Gerlach, W., Dixit, K. and Mäkinen, V., Compressed suffix tree - a basis for genome-scale sequence analysis. *Bioinformatics*, 23,5(2007), pages 629–630. URL `http://dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics23.html#ValimakiGDM07`.

Ros2011 van Rossum, G. and Drake, F. L., *The Python Language Reference Manual*. Network Theory Ltd., 2011.

WC1953 Watson, J. D. and Crick, F. H. C., Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, 171,4356(1953), pages 737–738.

Zen2010 Zeng, T. and Li, J., Maximization of negative correlations in time-course gene expression data for enhancing understanding of molecular pathways, 2010.