Implementing Computational Statistical Methods in a Scalable Individual-Tree Simulator

Kristoffer Paro

Master of Science Thesis
Supervisor: Jan Westerholm
Department of Information Technologies
Åbo Akademi University
April 2010

ABSTRACT

This thesis describes a set of computational statistical methods that were implemented in the scalable individual-tree forest simulator SPATE-HPC. Firstly, an overview of the simulator is provided. The thesis then describes the underlying theories behind the computational statistical methods and presents the corresponding program implementations. The implemented methods consist of an efficient, parallel method of computing spatially correlated random numbers, a method of transforming tree characteristics to adhere to an arbitrary distribution, a method of estimating tree trunk volumes and wood material yields, and a method of simulating low and selective thinning using Weibull distribution curves.

The numerical accuracy and the performance of the implementation for correlated random numbers were tested with extensive series of simulations. Furthermore, a separate case study on the functionality and synergy of the other implemented methods was conducted. The case study imitated a real-world scenario by simulating two different harvest schemes applied to the same forest area.

The methods were successfully implemented according to specified requirements. The implementation of correlated random numbers produced output of high accuracy in an efficient and scalable way. The case study showed that the other implemented methods behave as intended and deliver usable results.

Keywords: spatial correlation, distribution transformation, timber assortments, thinning methods, parallel programming

CONTENTS

LIST OF FIGURES

GLOSSARY

Assortment

A particular category of wood raw material obtained by cutting felled trees into logs.

Basal area

The cross-section of a tree's trunk measured at breast height.

Breast height

The height position 1.3 metres above the ground.

Cholesky decomposition

Also called Cholesky factorisation or matrix square root—a method of obtaining the matrix $\boldsymbol{L}$ given $\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$.

Clear cut

A final harvest in which most or all trees are cut down at the same time.

Gaussian random number

A random number drawn from a Gaussian (normal) probability distribution, denoted by $x \sim \mathcal{N}(\mu, \sigma^2)$, where μ is the mean and σ^2 is the variance.

Growing stock

The combined trunk volume of all living trees.

Mark

An inherent tree characteristic, such as diameter or height.

Poisson random number

A random number drawn from a Poisson probability distribution, denoted by $x \sim \mathrm{Pois}(\lambda)$, where λ is the mean. A Poisson distribution is a discrete distribution

describing the probability of a number of events occuring within a fixed time interval, where the events are independent and occur with a known average rate.

Root mean square error

A statistical measure of the magnitude of the difference between the observed and the predicted values, calculated as $\mathrm{RMSE}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \sqrt{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2/n}$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ are the observed values and $\hat{\boldsymbol{x}} = (\hat{x}_1, \ldots, \hat{x}_n)$ are the predicted values.

Sapling

A small tree, typically shorter than the breast height.

Seedling

A young plant raised from a seed.

Silviculture

The theory and practice of modifying and managing forests in order to control growth, composition, development and health of trees.

Stand

A group of trees that grow under similar conditions. Within this thesis the word stand is often used to refer to all simulated trees.

Superlinearity

A superlinear speedup is a phenomenon in parallel computing when a doubling of the number of processors results in a speedup of more than 2.

Thinning

A method in which trees are selectively cut, typically in order to improve overall forest growth and reduce tree mortality.

Uniform random number

A random number drawn from a continuous, uniform probability distribution, denoted by $x \sim U(a, b)$, where $[a, b]$ is the support.

Wall clock time

The real-world time it takes a computer to perform a certain task, measured from start to finish.

1 INTRODUCTION

Forests cover a significant part of Earth's land area and constitute a valuable natural resource. The wood material from trees can be used for a wide range of different products and purposes, such as structural material for construction, pulp for paper production, and fuel for bio-energy. Fluctuating market demands for different wood materials and new environmental legislations force forest owners to revise their harvesting practices. Because forests recover rather slowly, it can be deemed crucial that they be treated and harvested in a way that is both environmentally friendly and profitable for forest industries. As an alternative to conducting large-scale empirical studies on forest growth, computer simulations can be viewed as a viable method of obtaining statistical data on tree populations and wood yield volumes for both short- and long-term forest management.

Computers become progressively more effective with each generation, and the current trend is to incorporate an increasing number of processing units into a single computer instead of simply increasing the computer's clock frequency. This is a trend that is visible both in personal computers as well as on large supercomputer grids [1]. An essential aspect of developing computationally heavy software, such as a large-scale forest simulator, is thus to ensure that the software is able to fully utilise the computational power of parallel computer systems.

1.1 SUSWOOD project

The SUSWOOD (Sustainable and Environmental Friendly Wood Material Production for Future Industrial Needs) project is part of the research programme on Sustainable Production and Products (KETJU), financed by the Academy of Finland [2]. The project was carried out in the years 2007 to 2009.

The SUSWOOD project examined the potentials of alternative forest management systems to meet the needs for sustainable production of wood raw material as well as

maintaining forest ecosystem functions. It was a joint project carried out by a consortium consisting of the Department of Information Technologies at Åbo Akademi University, Mekrijärvi Research Station and the Department of Geography at the University of Joensuu, present day University of Eastern Finland.

The objective of the project partner at the Department of Information Technologies at Åbo Akademi University was to design, implement, test, and optimise a stochastic forest dynamics simulator for individual tree populations. Important aspects of the development of the simulator were to obtain highly efficient program code and software that scale well even on computer systems consisting of up to thousands of processors. When finished, the simulator was to be released as *open source* software—free for everyone to improve, specialise, and utilise in research, industry and education.

1.2 SPATE-HPC

The original version of the forest simulator, called SPATE (Spatio-Temporal Stand Simulator), was developed by Lin [3]. The program, however, was unable to scale linearly with the problem size and could not take advantage of multi-processor systems. It was therefore decided that the simulator should be redesigned as a highly efficient, parallel program that would be able to take advantage of the computational power of supercomputers with thousands of processors, in order to be able to simulate very large forest areas.

Because the original code, which was written in the Fortran programming language, was deemed too unstructured to be parallelised, the program was completely rewritten. The first version of the new simulator was programmed in C++ in an object-oriented fashion by Södergård [4], and it was subsequently parallelised using MPI (Message Passing Interface) [5] by Schöring [6]. The current version of the simulator is called SPATE-HPC, where HPC stands for *High Performance Computing*.

1.3 Purpose of this thesis

The purpose of this thesis is to describe a set of computational statistical methods that were implemented in SPATE-HPC since the work of Södergård and Schöring in order for the simulator to fulfil its design requirements and become a fully-functional software product. The computational statistical methods presented herein are the fol-

lowing: an efficient computation of spatially correlated random numbers, a distribution transformation of tree characteristics, a trunk and assortment volume estimation, and the simulation of low and selective thinning methods. The theories behind the methods are discussed, and their implementations are presented. Furthermore, the implementations are tested with two separate sets of simulations. Firstly, the implementation for correlated random numbers is tested with independent test cases, where the numerical accuracy and the performance are analysed. Secondly, the other implemented methods are tested by conducting a case study arranged to mimic a real-world forest growth and harvest scenario.

1.4 Thesis structure

An overview of the simulator is provided in chapter 2. The fundamental structure of the simulator—such as the different simulation phases, the growth models, and how trees interact—is introduced. A method for generating spatially correlated random numbers in an efficient and scalable way is presented in chapter 3. Chapter 4 describes how tree characteristics of artificially generated trees are transformed to adhere to a specified tree distribution. In chapter 5, formulae for estimating tree trunk volumes and algorithms for computing assortment- and species-specific yield volumes are explained. Chapter 6 describes the low and selective thinning methods, which use Weibull distribution approximations and stochastic processes to simulate thinning procedures. A case study of the implemented methods where two different management schemes are simulated is presented in chapter 7. The thesis concludes with a summary of the main results in chapter 8.

2 OVERVIEW OF THE SIMULATOR

This chapter describes the individual-tree simulator SPATE-HPC. Topics covered include how trees are represented, how the different phases of the simulation process are executed, the statistical models behind the simulation, and how the simulator reads input parameters and produces output data from the simulation.

2.1 Simulation phases

SPATE-HPC simulates the stand development in time as an iterative process consisting of a series of time steps. The length of a time step is specified by the user, and it may be one or several years. The user may also specify the number of time steps the simulator should run, which hence defines the number of years that are to be simulated.

A time step consists of four independent phases: growth, mortality, reproduction and management. The growth phase governs the growth of pre-existing trees, the mortality phase controls the death of trees, the reproduction phase creates new seedlings in uninhabited soil, and the management phase applies harvesting and thinning methods to the forest. The growth, the mortality and the reproduction are separate phases calculated independently of one another, and the results of the phases are committed after all of the phases have been completed. The management phase is performed as the last operation of a time step after the changes introduced by the growth, mortality and reproduction phases have been committed.

2.2 Representation of a tree

A tree in SPATE-HPC is represented as an individual object with a collection of attributes. It has a position with x- and y-coordinates and a set of inherent characteristics, called *marks*. The two marks defined in the current version of SPATE-HPC are the

tree's *diameter at breast height* (DBH) and its *height*. The DBH is the diameter measured at a point 1.3 metres above the ground, and the height is the tree's total height from the ground to the top.

Apart from the coordinates and the marks, a tree has an attribute indicating its species, it has an age, and it has an initial heterogeneity. The species can be of type Scots pine, Norway spruce, birch, etc., the age is simply the tree's age measured in years, and the initial heterogeneity is a Gaussian random number assigned to the tree upon its creation, which can be used as a component in the tree growth models to simulate natural uncontrolled variations in tree properties.

2.3 Spatial dependencies between trees

SPATE-HPC incorporates two types of spatial dependencies between neighbouring trees. Firstly, trees in close proximity are considered to compete for shared resources, such as water and sunlight. Although SPATE-HPC has no intrinsic concept of such resources, the actual results of the competition can be modelled, as long as it is known how the internal structure of a stand influences tree growth. It is for instance possible to model deficient growth due to overcrowded stands, so that one can observe the beneficial effects of periodically applied thinning. Secondly, SPATE-HPC includes random effects, called *environmental effects*, which may represent e.g. drought and insect infestation. The environmental effects affect trees locally; trees located closer to each other are more likely to be affected in a similar way than trees farther apart.

The neighbouring trees that are considered *competitors* to a given tree are those located within the *competition radius*. A tree's competitors may directly affect its growth and mortality. Figure 2.1 shows a sample stand of competing trees.

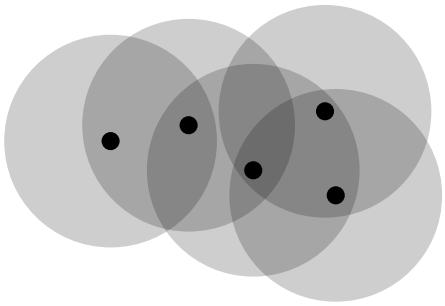


Figure 2.1: A sample stand. The black dots represent trees and the large, grey circles demarcate the competition area formed by the competition radii.

2.4 Compartments

SPATE-HPC employs the concept of compartments. A compartment is a forest area defined by one or several polygons. The size and shape of the polygons that constitute a compartment may be determined by ownership conditions or land features, such as rivers and lakes. Several polygons might be needed to represent a compartment if it is bisected by e.g. a road. Different compartments may contain different types of forest, that is, characteristics such as species composition, stand density and average tree size may vary between compartments. Figure 2.2 shows a forest area that has been divided into separate compartments.

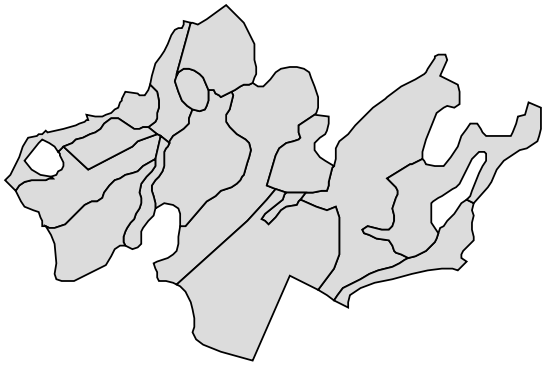


Figure 2.2: A sample forest area divided into compartments.

Management methods, i.e. harvesting and thinning, are applied individually to compartments. Different compartments may employ separate management schemes, and managements may be carried out at different times in different compartments.

However, compartment borders do not affect tree interactions; trees on different sides of a compartment border interact as if the border did not exist. When a simulation has finished, statistics are generated separately for each individual compartment.

2.5 Growth models

The fundamental calculations in the simulator governing growth, mortality and reproduction are based on generalised linear mixed models by Lin [3]. The models follow the pattern

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \alpha\boldsymbol{\tau} + \gamma\boldsymbol{\xi} + \boldsymbol{\varepsilon} \tag{2.1}$$

where $\boldsymbol{y}$ is a column vector of length n representing one type of mark (characteristic) for n trees, $\boldsymbol{X}$ is an $n \times p$ design matrix, $\boldsymbol{\beta}$ is a column vector of p regression coefficients, $\boldsymbol{\tau}$ is a random column vector of initial heterogeneity for n trees, $\boldsymbol{\xi}$ is a random column vector of n environmental effects, α and γ are scale parameters for $\boldsymbol{\tau}$ and $\boldsymbol{\xi}$, respectively, and $\boldsymbol{\varepsilon}$ is a column vector of n random numbers, representing an error term. Formula (2.1) can be rewritten to represent a mark of an individual tree i, in which case it takes the form

$$y_i = \boldsymbol{x}_i\boldsymbol{\beta} + \alpha\tau_i + \gamma\xi_i + \varepsilon \tag{2.2}$$

where $\boldsymbol{x}_i$ is the i-th row of the design matrix $\boldsymbol{X}$. The separate phases growth, mortality and reproduction may use different sets of values for the coefficients $\boldsymbol{\beta}$, α and γ.

The values of the design matrix $\boldsymbol{X}$ are calculated from *competition indices*. Each row of $\boldsymbol{X}$ consists of competition index values that are unique to the corresponding tree and vary from one time step to the next.

The environmental effects vector $\boldsymbol{\xi}$ consists of spatially correlated random numbers and represent random effects that affect trees in a local manner. The correlated random numbers are generated in such a way that trees located closer to one another are more strongly correlated, whereas trees farther apart are more weakly correlated. Correlated random numbers for environmental effects are discussed further in chapter 3.

The initial heterogeneity value τ of a tree is a normally distributed random number that is generated once when the tree is created and kept constant thereafter. The error term value ε, on the other hand, is a normally distributed random value that is redrawn at each instance it is used.

2.5.1 Competition indices

Competition indices are used to model the competition for shared resources among neighbouring trees. Competition indices are calculated individually for each tree. They depend upon the target tree's characteristics, as well as the tree's nearby neighbours that reside within the predetermined competition radius. Examples of competition indices are the target tree's diameter, the tree density within the competition area, the average distance to competitors, and the sum of all competitors' diameters.

2.5.2 Secondary-level models

The initial version of SPATE-HPC used a model restricted to formula (2.1) in the sense that the regression coefficients $\boldsymbol{\beta}$ were equal for all trees. The functionality was subsequently extended to allow for a secondary level of variable coefficients. The secondary-level model functionality allows for the fixed-effects regression coefficients to be separate for different categories of trees, such as trees of different species or trees growing under different conditions. An element β_j of $\boldsymbol{\beta}$ in formula (2.2) can be expressed as the secondary-level model

$$\beta_j = \boldsymbol{x}^* \boldsymbol{\beta}_j^* + \varepsilon^* \tag{2.3}$$

where $\boldsymbol{x}^*$ is a row vector of m secondary-level variables, $\boldsymbol{\beta}_j^*$ is a column vector of m coefficients, and ε^* is a normally distributed random number, representing an error term. A variable in the secondary-level model can for example be the tree's age class, its size class, or a management- or species-specific constant. As with the top-level model, the coefficients $\boldsymbol{\beta}_j^*$ may be different in the different growth phases.

2.5.3 Generating an initial stand

If empirical tree data are not available for the initial stand, trees can be generated automatically according to a set of criteria. The number of generated trees is determined by a Poisson random number with a specified number of trees per hectare as the expected value. Moreover, a point process is used to position the newly created trees. A point process is a random element whose values are points on a set [7]. In this application, the points are position coordinates in a two-dimensional Euclidean space.

After the trees have been placed, they are assigned marks according to a model of

the form presented in formula (2.2). The model is first applied to generate diameter marks and then height marks. This means that the height marks can be made dependent on the generated diameter marks. The marks may furthermore be transformed to adhere to a user-defined distribution. The process of transforming tree marks is described further in chapter 4.

2.5.4 Growth

Formula (2.2) is used to represent tree growth on a logarithmic scale. The size of tree i at time step t is determined by the formula

$$\ln(y_i^t) = \ln(y_i^{t-1}) + \boldsymbol{x}_i^t\boldsymbol{\beta} + \alpha\tau_i + \gamma\xi_i^t + \varepsilon \tag{2.4}$$

where y denotes a tree mark, i.e. the diameter or the height, and the superscript t denotes values at time step t. By observing exponentiation rules, formula (2.4) can be rewritten in the form

$$y_i^t = y_i^{t-1} \cdot \exp(\boldsymbol{x}_i^t\boldsymbol{\beta} + \alpha\tau_i + \gamma\xi_i^t + \varepsilon). \tag{2.5}$$

The factor $\exp(\boldsymbol{x}_i^t\boldsymbol{\beta} + \alpha\tau_i + \gamma\xi_i^t + \varepsilon)$ is hence the growth factor with respect to the previous time step.

2.5.5 Mortality

SPATE-HPC incorporates two types of tree mortality: a regular mortality and an irregular, random mortality. The regular mortality is due to deficient growth, which can occur with old trees and within stands with high competition. A tree experiences deficient growth if its growth factor at any given time step is less than a predefined threshold. If that is the case, the tree is marked as dead. If the regular mortality threshold is set to a value lower than 1, the trees are allowed to shrink.

The irregular mortality is due to forest damage and unexplained causes and is modelled with a logistic regression model based on formula (2.2). An individual tree i at time step t is killed according to the probability π_i^t, where π_i^t is calculated as

$$\pi_i^t = \frac{1}{1 + \exp(-\boldsymbol{x}_i^t\boldsymbol{\beta} - \alpha\tau_i - \gamma\xi_i^t - \varepsilon)}. \tag{2.6}$$

2.5.6 *Reproduction*

The reproduction phase is similar to the initial stand generation phase. The number of new seedlings in a compartment is determined by a Poisson random number with the expected value calculated according to a regeneration model, i.e.

$$n \sim \mathrm{Pois}(a_0 + a_1 w_1 + a_2 w_2 + \ldots + \varepsilon) \tag{2.7}$$

where n is the number of new trees, a_i are regeneration coefficients, w_i are compartment-specific variables—such as the number of pre-existing trees, the growing stock, or the total basal area—and ε is a random number, representing an error term. The new trees are then placed into each compartment as a homogeneous Poisson point process.

Lastly, the new trees are assigned marks according to formula (2.2). The competition indices are calculated from already existing trees, whereas the environmental effects are calculated only from newly created trees. When the marks have been calculated, they are transformed to a target distribution according to the transformation procedure described in chapter 4.

2.6 Multiple replications

SPATE-HPC employs a *Monte Carlo* method [8] to obtain statistically reliable results. The simulator can perform multiple simulations with identical initial conditions and model parameters but with differently drawn random numbers. These repeated simulations are referred to as *replications* and constitute a Monte Carlo method for obtaining statistical results within a given confidence interval. The user may choose the number of replications the simulator should run, and the simulator produces a statistics summary with average tree characteristics according to a 95% confidence interval.

2.7 Input data

The simulator is configured via parameters given in an input file in the XML (Extensible Markup Language) format [9]. Here, virtually all parameters governing the simulation are given. These include the number of time steps and replications to simulate, the species, the compartments, the management methods, etc. All coefficients

and variables of the models governing growth, mortality and reproduction are also specified in the XML file.

The XML input file may furthermore reference external files. The polygons that constitute the compartments are supplied in polygon files. A polygon file may contain several polygons. Other types of external data the XML file may reference include empirical data for the initial stand and for mark transformations. If the initial stand is to be generated, empirical data can be used as a target distribution. If the initial stand is to be loaded from a tree data file, the tree data file is provided as an external file referenced to by the XML file.

2.8 Output data

The simulator produces different types of output both during and after a simulation. When all replications have been simulated, a separate statistics file for each compartment is generated. The statistics files contain information about the species composition, the average tree size, the stand development throughout the simulation, as well as assortment- and species-specific volumes for both the growing stock and the harvest yields.

Additionally, tabular data files containing information about each tree, such as x- and y-coordinates, diameter, height, age, etc., can be saved to disk. The tree data files are saved at the end of the simulation and, optionally, after each time step as well. These files can be used as input for external analysis software or be used as the initial stand in subsequent simulations.

Each processor continuously writes a log file during the simulation. The log files provide information about each simulation step as it is carried out. Examples of such information are memory consumption, wall clock time and the number of trees included in a particular operation. Error and warning descriptions are also written to the log files in case of an anomalous situation. Anomalous situations may occur if the user has specified invalid parameters, or if the parameters are such that the forest grows uncontrollably.

3 CORRELATED RANDOM NUMBERS

The environmental effects functionality of SPATE-HPC can be used in conjunction with the initial stand, growth, mortality and reproduction models based on formula (2.1). Environmental effects are denoted by $\boldsymbol{\xi}$, a vector of spatially correlated random numbers. The correlated random numbers are generated in such a way that trees located closer to one another are more strongly correlated, whereas trees farther apart are more weakly correlated or not correlated at all.

A commonly used method [10] for generating spatially correlated Gaussian random numbers is to first create a covariance matrix $\boldsymbol{\Sigma}_\xi$ of size $n \times n$, where n is the total number of trees and each element σ_{ij} of $\boldsymbol{\Sigma}_\xi$ represents the covariance between the two trees i and j. The covariance matrix $\boldsymbol{\Sigma}_\xi$ is decomposed using the *Cholesky decomposition* algorithm [11], producing a lower-triangular matrix

$$\boldsymbol{L}_\xi = \begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}$$

in such a way that

$$\boldsymbol{L}_\xi \boldsymbol{L}_\xi^{\mathrm{T}} = \boldsymbol{\Sigma}_\xi. \tag{3.1}$$

Finally, the correlated random numbers are obtained by multiplying the lower-triangular matrix by a column vector of uncorrelated Gaussian random numbers $\boldsymbol{z} = (z_1, z_2, \ldots, z_n)^{\mathrm{T}}$, $z_i \sim \mathcal{N}(0, 1)$:

$$\boldsymbol{\xi} = \boldsymbol{L}_\xi \boldsymbol{z}. \tag{3.2}$$

In SPATE-HPC, the covariance between the trees i and j is calculated from

$$\sigma_{ij} = \begin{cases} \exp\left(\theta_1 + \theta_2 d_{ij}\right) & \text{if } d_{ij} < r_c \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where d_{ij} is the distance between the trees, r_c is the competition radius and $\theta_1,\ \theta_2 < 0$ are parameters given by the user [3].

3.1 Limitations of the original implementation

The original implementation in SPATE-HPC to compute spatially correlated random numbers was programmed strictly according to formulae (3.1), (3.2) and (3.3); the program composed and maintained a large covariance matrix containing the variances and covariances of all trees in the entire simulation. This implementation imposes two serious problems. Firstly, since the covariance matrix is of the size $n \times n$, the matrix—and hence also the memory consumption—will grow quadratically with respect to the total number of trees. Secondly, the computation time of a Cholesky decomposition increases cubically with respect to the number of elements [12].

Suppose that each matrix element is a 64-bit double-precision floating-point number. The matrix' memory consumption can hence be calculated as $8n^2$ bytes. For a tiny forest of 1,000 trees, the matrix' memory consumption would only be a modest 7.6 MB, but for a forest containing 10,000 trees, the memory consumption would already grow to 763 MB. A forest containing 1,000,000 trees would consume the impractical amount of 7.5 TB. It is possible to reduce this memory requirement to some extent by taking into account the sparsity of the covariance matrix and also by discarding parts of the matrix that are no longer needed in the computation. Nonetheless, the Cholesky decomposition algorithm must still maintain a quadratic lower-triangular matrix during the calculation process. The quadratic scaling behaviour would persist.

Moreover, as shown by Santos and Chu [12], the lower bound on the number of computations needed for a full Cholesky decomposition is $\Omega(n^3)$. Apart from requiring immense amounts of memory, simulating large forests would hence quickly become very time-consuming as well.

There are algorithms for computing the Cholesky decomposition in parallel, as demonstrated by Gupta and Kumar [13] and Santos and Chu [12]. A parallel imple-

mentation of the Cholesky decomposition would certainly be an improvement over the sequential one. However, as long as the resulting matrix is allowed to grow quadratically, the total memory consumption will also grow quadratically. Even if the parallelisation of the problem were ideal, and each processor maintained its own part of the matrix, the memory used by each processor would still be proportional to n^2/p, where p is the number of processors. The problem of having a cubic computation time growth also persists in parallel implementations of the algorithm. According to Santos and Chu, the lower bound on the computation time for any parallelised Cholesky decomposition algorithm is $\Omega(n^3/p + n)$ [12].

Since SPATE-HPC should be able to simulate very large forest areas [2], possibly comprising millions of trees covering thousands of hectares, it can thus be concluded that the Cholesky decomposition is not a scalable, nor feasible way of computing correlated random numbers in this simulator.

3.2 An approximate, scalable algorithm

Instead of parallelising the original Cholesky algorithm, a different approach is to allow the resulting lower-triangular matrix to be only *approximately* equal to the mathematically correct one. The competition radius used in SPATE-HPC is typically much smaller than the length of the simulated forest in any given direction. Analysing formula (3.3) reveals that the covariance matrix $\mathbf{\Sigma}_\xi$ will be very sparse. Although trees farther apart than the competition radius have a mutual covariance of zero (and thus do not correlate directly), the computation of the correlated random number for one of the trees might depend on previous computations involving the other tree. This is an inherent property of the Cholesky decomposition.

A modified algorithm presented herein assumes that the covariance matrix is sparse and that the random-number coefficients in the lower-triangular matrix introduced between uncorrelated trees far apart become small. By neglecting the small coefficients, a *local* Cholesky decomposition can be performed for a particular tree by including only its nearby neighbours. The algorithm begins by generating and assigning an independent Gaussian random number to each tree, $\boldsymbol{z} = (z_1,\, z_2, \ldots,\, z_n)^{\mathrm{T}}$. Initially, all trees are considered unprocessed. The algorithm then iterates through the following steps until all trees have been processed.

1. An unprocessed tree is picked arbitrarily as the current target tree.

2. A local covariance matrix $\tilde{\boldsymbol{\Sigma}}_\xi$ is created such that the target tree is the last one to be included. Included before the target tree are its neighbours that have already been processed. Neighbours are all trees that are located within the distance $a \cdot r_c$ from the target tree, where $a \geq 1$, referred to as the *inclusion factor*, is a configurable parameter. For small values of a, the size of $\tilde{\boldsymbol{\Sigma}}_\xi$ will be very small as there are only a small number of neighbours compared to the total number of trees in the entire forest.

3. The local covariance matrix $\tilde{\boldsymbol{\Sigma}}_\xi$ is decomposed using the standard Cholesky decomposition procedure, i.e. $\tilde{\boldsymbol{L}}_\xi \tilde{\boldsymbol{L}}_\xi^{\mathrm{T}} = \tilde{\boldsymbol{\Sigma}}_\xi$.

4. The correlated random number for the target tree is calculated as $\tilde{\boldsymbol{l}}_{\xi_n} \boldsymbol{\zeta}$, where $\boldsymbol{\zeta}$ is a column vector of random numbers from $\boldsymbol{z}$, ordered in the same order as the corresponding trees appear in $\tilde{\boldsymbol{\Sigma}}_\xi$, and $\tilde{\boldsymbol{l}}_{\xi_n}$ is the last row of $\tilde{\boldsymbol{L}}_\xi$. Only the last row is needed, because the preceding rows correspond to trees whose correlated random numbers have already been calculated in earlier iterations.

5. The target tree is marked as processed.

The first tree picked by the algorithm will always be alone. This is in analogy with the full Cholesky decomposition, where the tree's resulting correlated random number will only depend upon the tree's initial random number itself, i.e. $\xi_1 = l_{11} z_1$. For the second tree, the correlated random number will be $\xi_2 = l_{21} z_1 + l_{22} z_2$ with both algorithms, assuming that trees 1 and 2 are neighbours. Suppose that the correlated random number for tree i is to be calculated. The result from the full Cholesky decomposition would be $\xi_i = (l_{i1}, \ldots, l_{ii})(z_1, \ldots, z_i)^{\mathrm{T}}$. If the preceding trees are not neighbours of tree i, then the values $l_{i1} \ldots l_{i,i-1}$ will by assumption be small and approximated to zero by the local Cholesky decomposition algorithm by simply not including them. By increasing the inclusion factor a, which governs the maximum distance to trees considered being neighbours, the approximation error of the end result is likely to decrease. If a is large enough to include all trees in the forest, the end result will be identical to a full Cholesky decomposition.

3.2.1 Parallelisation

The aforementioned approximate Cholesky decomposition algorithm is fairly straightforward to implement sequentially. Nevertheless, some additional difficulties arise

when adapting the algorithm to a parallel environment.

Some of the trees that are processed by the local Cholesky decomposition will stand close to a process' border. The trees will then possibly have neighbouring trees that reside within the boundaries of an adjacent neighbouring process. These trees that belong to another process but stand close enough to be neighbours of trees within the current process are referred to as *foreign* trees. Aside from the process' own trees the local Cholesky decomposition also needs the foreign trees that belong to the adjacent processes. The functionality of communicating foreign trees across process boundaries is already implemented in SPATE-HPC, since foreign trees are needed in order to calculate competition indices. The inter-process tree-communication function need only be extended to include trees a times farther away, where a is the inclusion factor.

A considerably more complicated issue with the parallelisation is the problem of computing the correlated random numbers in the correct order. Because the algorithm only includes already processed neighbours in the local covariance matrix, any individual process must know which of the near-border foreign trees have been processed and which ones have not. Each process also needs the corresponding independent random numbers (z) for the foreign trees it includes as neighbours.

One possible approach to this problem would be to have a continuous stream of inter-process communication, where adjacent processes are constantly notified of processed trees. However, having to continuously communicate between processes is something that is largely shunned in parallel programming, as computation is generally fast and communication is generally slow [14].

An alternative approach is to explicitly define an order in which the trees are to be processed. This way an individual process is able to deduce by itself whether foreign trees have been processed, based upon its own current state. For example, if each process starts at its bottom edge, the neighbouring foreign trees of the process directly below it will be guaranteed to be unprocessed. Similarly, if each process continues to process trees with increasing y-coordinates, at the time a process reaches its upper edge, the neighbouring foreign trees of the process directly above will have been processed.

When the forest area is divided amongst processes, every process' area is not necessarily of equal size. The area is first divided into columns, which are furthermore divided into rows. All processes in a particular column will therefore have areas of the same width. The implementation exploits this fact. The functionality of dividing the

forest area is implemented in the *ForestSplitter* class, and the procedure is explained in detail by Schöring [6].

One usable constraint imposed by the *ForestSplitter* class is that the length of the side of a process' area must be no shorter than twice the *limiting distance*. The limiting distance has hitherto been the competition radius, but can now be extended to the correlation distance multiplied by the inclusion factor, so that any individual process need not consider neighbouring trees farther away than to the midpoint of the adjacent processes.

Each process is split along a vertical line into two strips of equal width. A process begins by processing only the trees in the left strip, starting with its bottom trees and proceeding upwards to the top trees, as shown in figure 3.1. As the process proceeds along the left strip, all foreign trees to the left are guaranteed to be unprocessed. Only the trees in the left strip of the adjoining process directly above have been processed at the point the process reaches its topmost trees. The process then continues with the right strip, also starting with its bottom trees and proceeding upwards. This time, all foreign trees have been processed *except* the top trees of the adjoining process directly below. This is shown in figure 3.2.

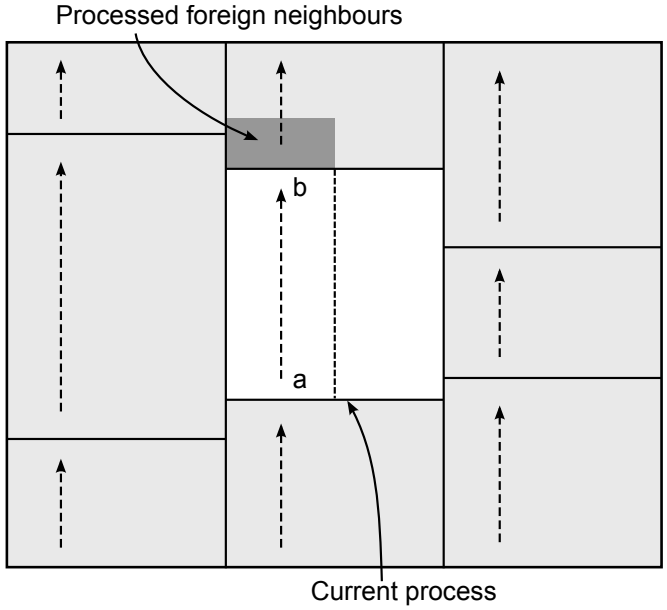


Figure 3.1: Trees in the left strip are processed from *a* to *b*. Adjacent processes are coloured light grey, and areas containing processed foreign neighbouring trees are coloured dark grey.

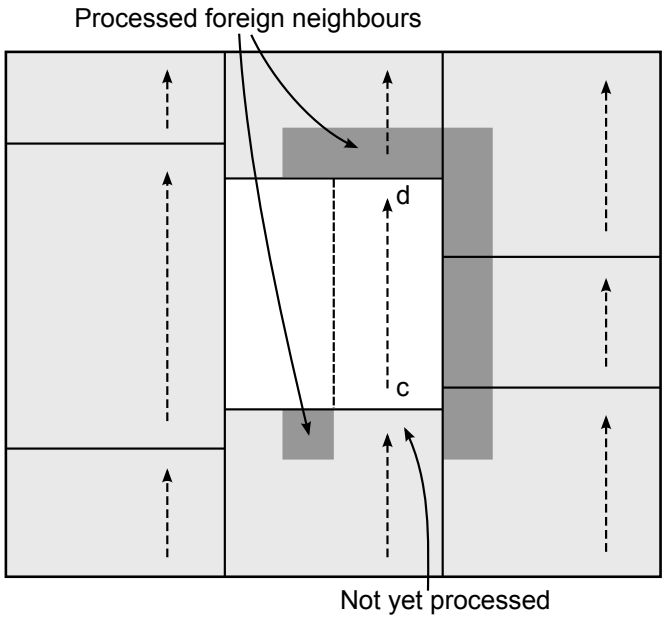


Figure 3.2: Trees in the right strip are processed from c to d. Adjacent processes are coloured light grey, and areas containing processed foreign neighbouring trees are coloured dark grey.

By processing trees in the aforementioned order, the only new inter-process communication that must be introduced constitutes the normally distributed random numbers associated with the trees. Because the bordering trees are always communicated at the beginning of each time step, the trees can be assigned random numbers already at this point and subsequently be distributed to adjacent processes along with the other tree data. Moreover, since the computation of correlated random numbers for different instances of environmental effects are unrelated operations, there must be one set of random numbers for each tree mark as well as for each simulation phase (e.g. growth and mortality) where environmental effects can be used.

The requirement that trees be processed in a certain order is readily solved by sorting them. The sorting and re-sorting of trees must be done whenever new trees have been generated. This is the case for generated trees in the initial stand as well as for new trees in the reproduction phase. Fortunately, because trees are accessed via an array of pointers (in the *TreeContainer* class), and it is sufficient to sort only the pointers, the sorting procedure becomes fairly lightweight.

Sorting the trees in each process in the aforementioned order solves the problem of not knowing whether a foreign tree has been processed. However, one problem still remains: knowing *in which order* previously processed neighbours have been pro-

cessed. Since foreign trees may originate from different adjacent processes, they may appear in any order once they have been collected from the adjacent processes. If a local Cholesky decomposition is performed with the preceding trees in a different order than the one in which they were processed in the neighbouring processes, the result will not be exactly the same. It is therefore important that the foreign neighbouring trees be included in exactly the same order as the one in which they were processed in the other processes.

The problem is solved by introducing relative process coordinates. The relative process coordinates for a tree are in the interval $[0, 1]$ and are calculated as

$$x_r = \frac{x - x_{p,0}}{x_{p,1} - x_{p,0}} \quad \text{and} \quad y_r = \frac{y - y_{p,0}}{y_{p,1} - y_{p,0}} \tag{3.4}$$

where (x, y) is the tree's actual coordinates, and $(x_{p,0}, y_{p,0})$ and $(x_{p,1}, y_{p,1})$ are the process' maximum and minimum coordinates, respectively. The minimum and maximum values of the relative coordinates are the same regardless of the shape of the process' area, and the left and the right strip are separated by the relative x-value of 0.5. The relative coordinates are illustrated in figure 3.3.

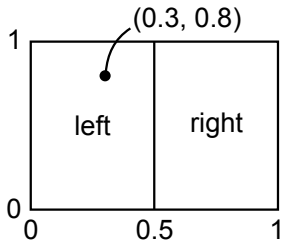


Figure 3.3: Relative process coordinates. A sample tree has the relative coordinates (0.3, 0.8).

The relative process coordinates of a tree reveal exactly in which stage of the procedure the tree has been processed. A small y-value and an x-value less than 0.5 indicate that the tree was processed early. Likewise, a large y-value and an x-value greater than 0.5 indicate that the tree was among the last ones to be processed. This information can be made available to each process by assigning the relative process coordinates to trees already in the beginning and then transferring them along with the other tree data to the adjoining processes. Because relative process coordinates indicate the relative order in which trees have been processed, a list of neighbouring trees—both native and foreign—can be sorted with respect to these relative coordinates. The trees in the

sorted list will then appear in the order they were processed.

3.3 Implementation

The core functionality for calculating correlated random numbers with the local Cholesky decomposition algorithm is implemented in the class *LocalCorrelation*. The class is used by the *Forest* class during the growth, mortality and reproduction phases and by the *InitialCompartmentGenerator* class when tree marks for the initial stand are generated. The class *TreeSorter* is an auxiliary class that provides methods for sorting trees so that all processes may traverse the trees in the same order. Figure 3.4 shows a class diagram of the classes used for the local Cholesky decomposition.

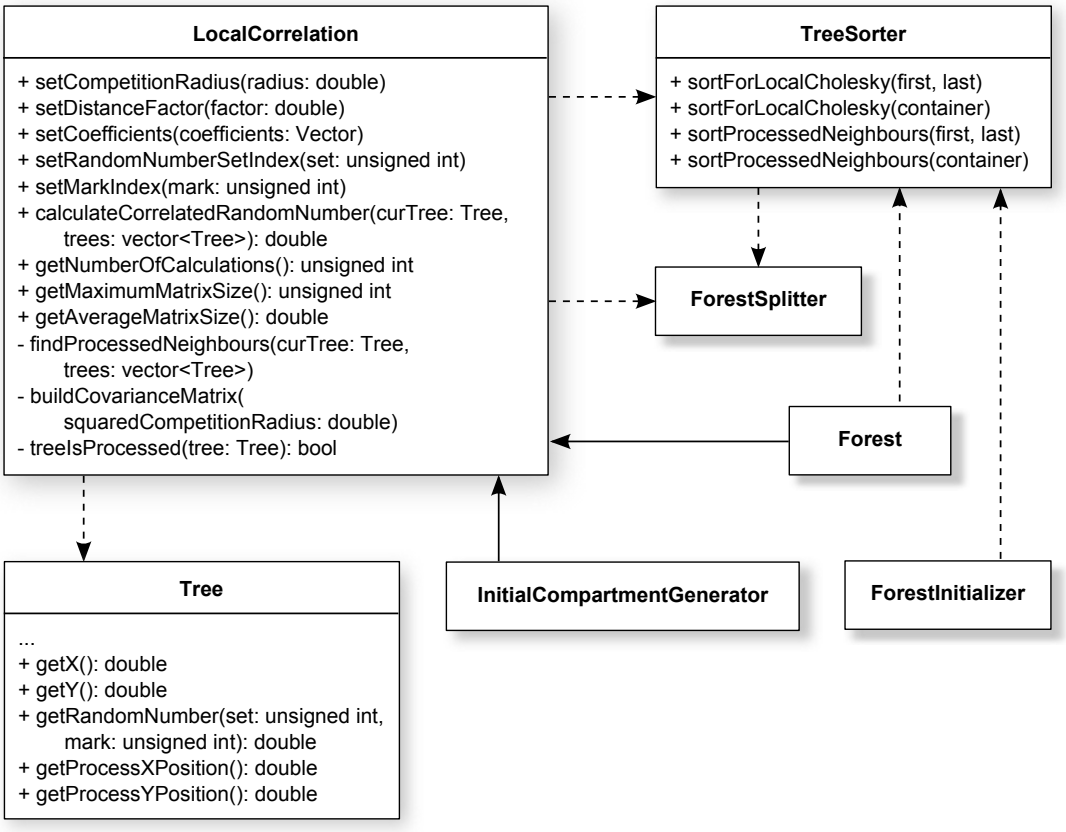


Figure 3.4: Class diagram of the classes used for the local Cholesky decomposition.

3.3.1 *LocalCorrelation*

An instance of the *LocalCorrelation* class is used to compute a specific set of correlated random numbers for a group of trees. Correlated random numbers in the growth, mortality and reproduction phases are all independent sets and are computed separately with different instances of the class.

A correlated random number for a specific tree is computed by calling the *computeCorrelatedRandomNumber* method, which implements the algorithm described in section 3.2. The method begins by finding processed neighbours by traversing neighbouring trees and comparing them to an internally maintained set of already-processed trees. If the trees are members of the internal set, they are included in the computation process. The trees are sorted by calling the method *sortProcessedNeighbours* of the *TreeSorter* class, and a local covariance matrix is built using the covariance function and is subsequently decomposed. The uncorrelated random numbers are fetched by calling *getRandomNumber* on the corresponding trees and are then multiplied by the matrix' last row in order to obtain the final correlated random number for the target tree.

3.3.2 *TreeSorter*

The *TreeSorter* class contains the methods *sortForLocalCholesky* and *sortProcessedNeighbours*, which are used to sort a collection of trees according to their relative process coordinates. The relative process coordinates are obtained by calling the methods *getProcessXPosition* and *getProcessYPosition* on a *Tree* instance.

The method *sortForLocalCholesky* assigns relative process coordinates to the trees as described in section 3.2.1 and sorts them according to the y-coordinate so that trees in the left strip appear before trees in the right strip. The method is called at the beginning of a time step or when a new set of trees are created, such as in the initial stand generation and the reproduction phase, in order to arrange the trees so that all processes traverse them in the same order.

The *sortProcessedNeighbours* method is called from within a *LocalCorrelation* instance on a set of neighbouring trees in order to sort them according to their relative process coordinates. When a group of neighbouring trees are collected near process borders the same trees may appear in a different order on other processors. By sorting the trees with this method it is ensured that they appear in the same order in all

processes when included as neighbours in the local covariance matrix.

3.3.3 *Forest*

A sequence diagram illustrating how the classes are used in the growth phase is presented in figure 3.5. At the beginning of a time step, which is represented by the *simulateIteration* method call to the *Forest* class, all trees are assigned uncorrelated random numbers with the *assignRandomNumbers* method call. Each tree is assigned four random numbers, one for each mark (diameter and height) and each phase (growth and mortality). The trees are then sorted according to their positions with the *sortForLocalCholesky* method and trees located near process borders are transferred to adjacent processes with the *transferForeignTrees* method.

The first phase is the growth phase, which is performed by the *calculateGrowth* method. The method iterates through all trees separately for each mark and uses an instance of the *LocalCorrelation* class to compute the correlated random numbers. When all correlated random numbers have been computed, the growth of each tree is calculated with the growth model where the correlated random numbers are used for the environmental effects. The mortality phase, which follows the growth phase, is implemented in the *calculateMortality* method and is analogous to the growth phase regarding the correlated random numbers.

The reproduction phase is slightly different from the growth and the mortality phases, since here the correlated random numbers are not computed for all trees in the entire forest but only for the newly created ones. A sequence diagram showing the reproduction phase is presented in figure 3.6. First the positions of the new trees are computed separately for each compartment. Then the new trees are assigned uncorrelated random numbers with the *assignRandomNumbers* method, sorted with the *sortForLocalCholesky* method, and transferred to adjoining processes with the *transferForeignTrees* method. Finally, correlated random numbers are computed for each mark of each new tree with the *calculateCorrelatedRandomNumber* method and are subsequently used for the environmental effects in the reproduction model.

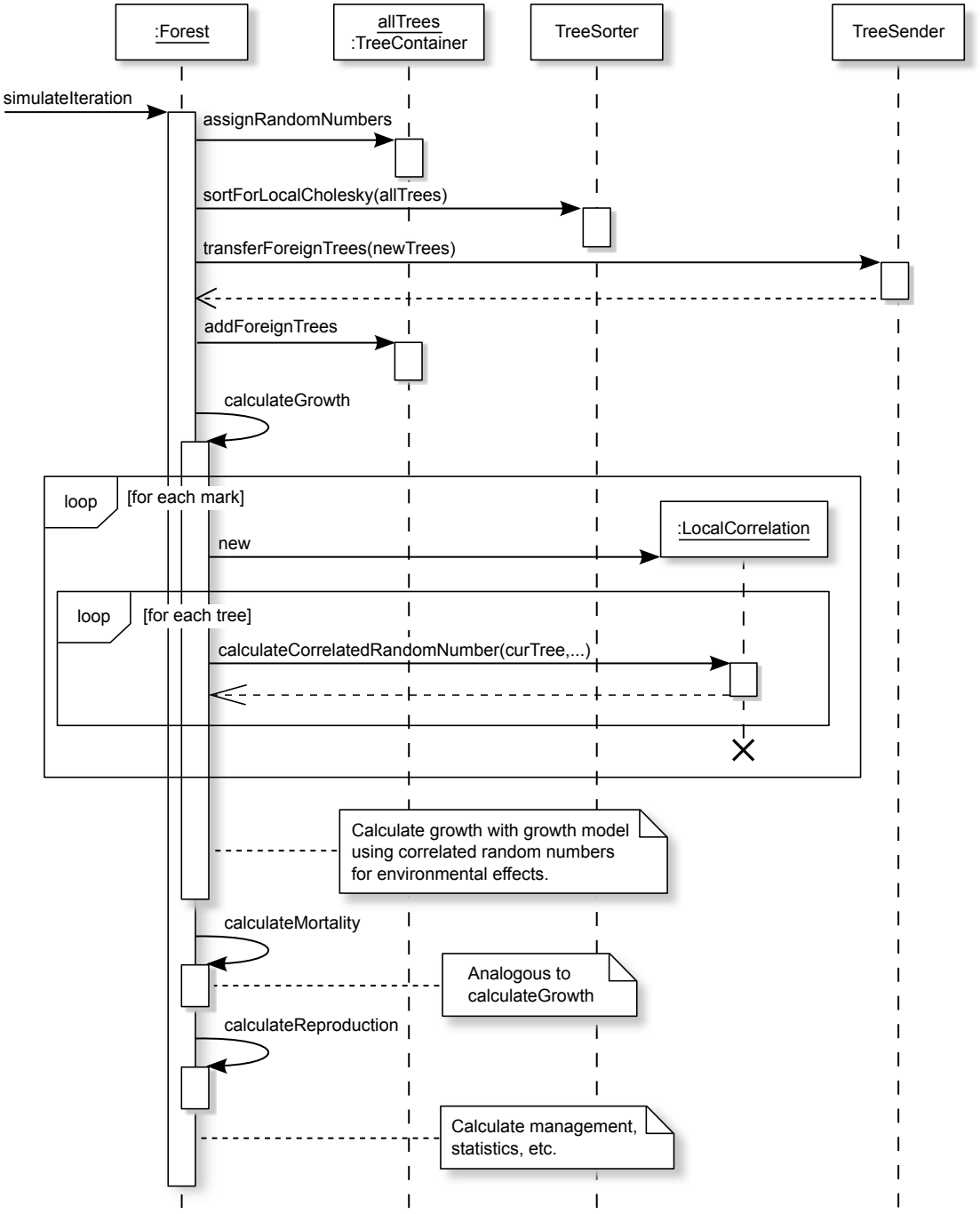


Figure 3.5: A sequence diagram illustrating how correlated random numbers are computed in the growth phase.

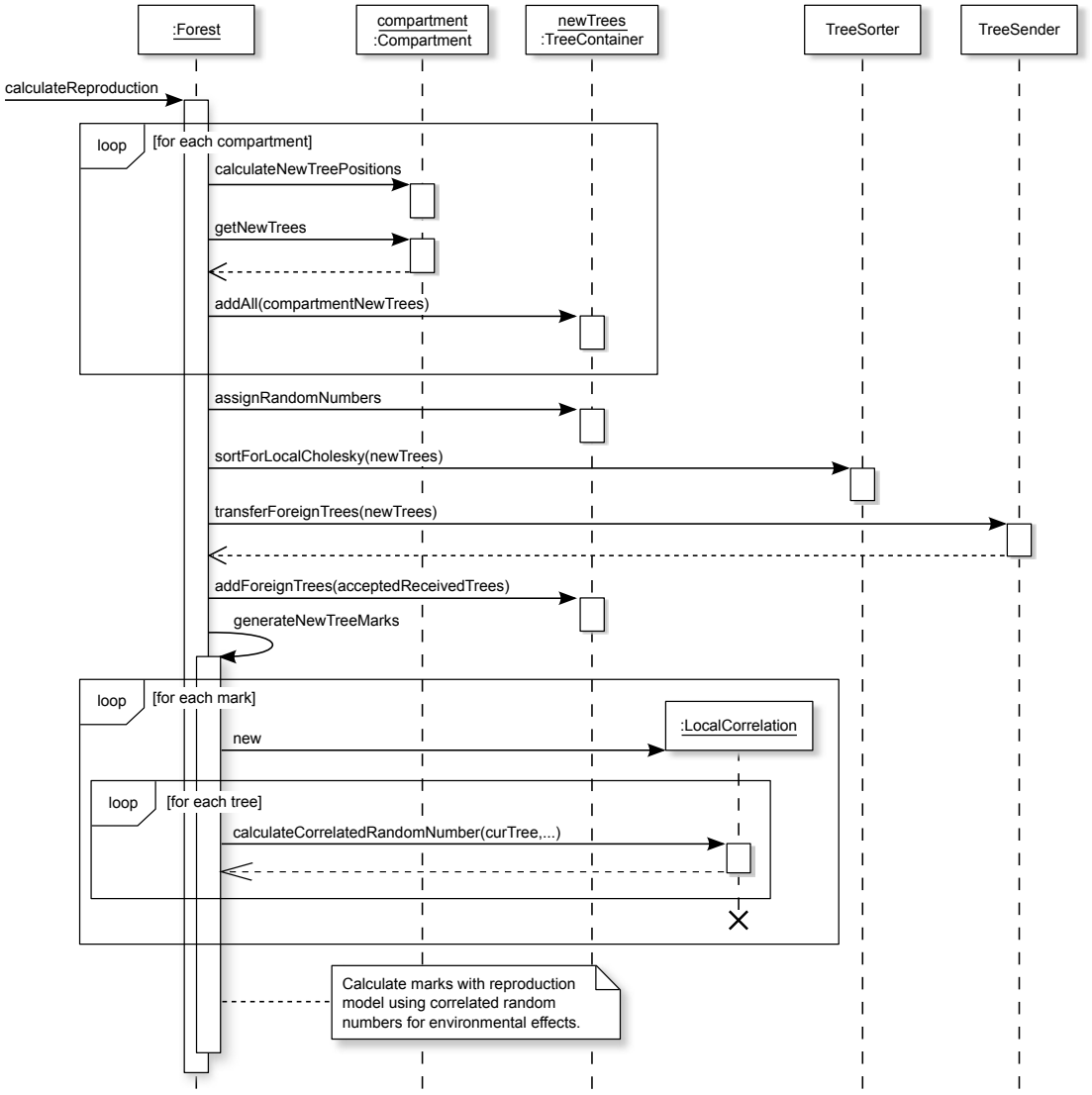


Figure 3.6: A sequence diagram illustrating how correlated random numbers are computed in the reproduction phase.

3.4 Numerical accuracy

Since the local Cholesky decomposition method only approximates the conventional method, the resulting correlated random numbers will not have exactly the same correlation structure when the local algorithm is used compared with the global one. In order to assess the numerical accuracy of the local Cholesky decomposition, two different test cases were devised: a direct comparison and a correlation structure analysis.

3.4.1 Direct comparison

The first test case directly measured the approximation error induced by the local Cholesky decomposition. This was done by running two identical simulations using both the global and the local Cholesky implementation in such a way that exactly the same sequence of uncorrelated random numbers were drawn on both runs. Comparable sequences of random numbers were obtained using the *comparison seeding* debugging functionality [6], which artificially seeds the random number generator.

In the test case a simulation was run on a square area of one hectare containing approximately 3,000 trees. The simulation comprised only the initial stand generation phase, where the trees' diameters were initialised to the environmental effects only. The parameters for the covariance function (3.3) were set to $\theta_1 = 0$, $\theta_2 = -0.33$, and the competition radius was 15 metres. No distribution transformations were applied. This way, when the simulation had finished, the correlated random numbers were obtainable as the diameters of the trees. In order to analyse the effects of the inclusion factor on the approximation error, varying factors between one and three were tested.

Due to the comparison seeding functionality, both the simulation using the global decomposition algorithm and the one using the local decomposition algorithm yielded exactly the same trees on exactly the same coordinates using exactly the same uncorrelated random numbers. The only thing that differed was the computed correlated random numbers and hence the resulting tree diameters, which could be compared directly. The results of the simulations are presented in figure 3.7 and table 3.1. The figure and the table show the root mean square error and the maximum approximation error with different inclusion factors, while the table also lists the average and maximum sizes of the matrices used for the Cholesky decomposition as well as the simulation wall clock time and peak memory usage. The results show a relatively small approximation error that decreases significantly with larger inclusion factors. A larger

inclusion factor, however, increases the simulation wall clock time and the memory usage.

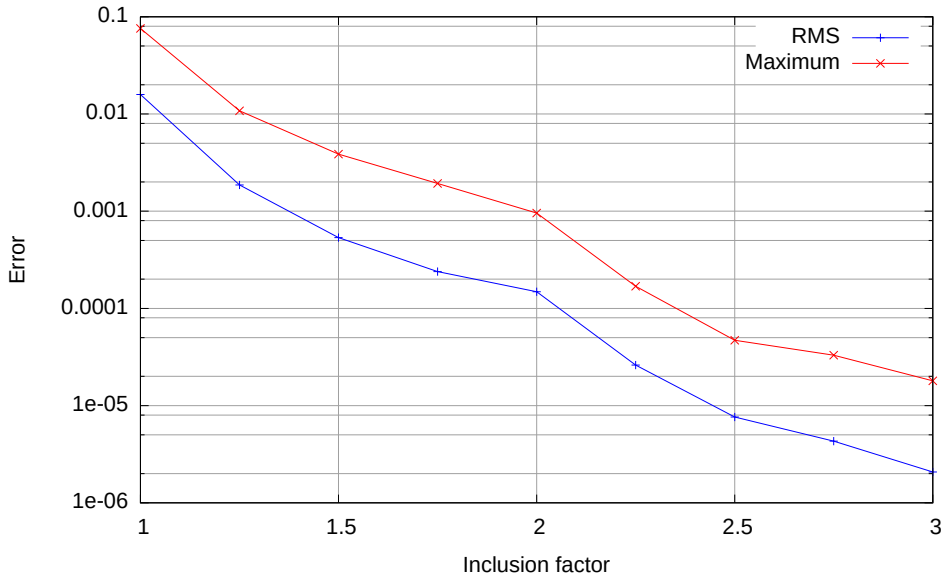


Figure 3.7: The root mean square (RMS) and maximum error of the computed correlated random numbers obtained with different inclusion factors.

Table 3.1: Simulation outcome with varying inclusion factor

Inclusion factor	1.0	1.5	2.0	2.5	3.0
Root mean square error	0.016	$5.4 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	$7.6 \cdot 10^{-6}$	$2.1 \cdot 10^{-6}$
Maximum error	0.076	$3.9 \cdot 10^{-3}$	$9.6 \cdot 10^{-4}$	$4.7 \cdot 10^{-5}$	$1.8 \cdot 10^{-5}$
Cholesky matrix size ($n \times n$)					
Average	95.0	198.8	328.7	476.7	634.3
Maximum	243	507	876	1370	1948
Wall clock time (s)	2.0	10.3	38.4	103	230
Peak memory usage (MB)	9.1	10.8	14.3	22.3	37.6

3.4.2 Correlation structure analysis

The second test case studied the applicability of the local Cholesky decomposition algorithm on a larger scale by running a series of simulations and calculating the spatial correlation of the resulting random numbers. The simulation comprised a 36 ha square

area containing approximately 108,000 trees, and 100 replications were simulated. Like the first test case only the initial stand was generated with the trees' diameters being the environmental effects. Unlike the first test case, however, no artificial seeding of the random number generator was employed, that is, all replications produced different sequences of uncorrelated random numbers.

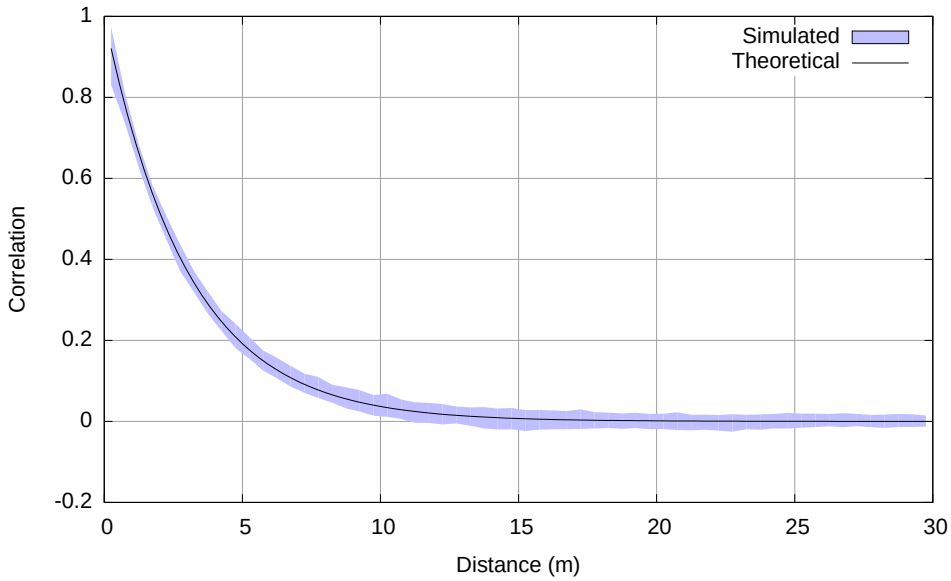


Figure 3.8: A correlogram showing the measured correlation from 100 replications compared to the theoretical correlation.

The spatial correlation was measured with Moran's I formula [15] at increasing lag distances [16]. The spatial correlation $I(h)$ for lag h is given by

$$I(h) = \frac{n}{\sum_i \sum_j w_{ij}(h)} \frac{\sum_i \sum_j w_{ij}(h)(\xi_i - \bar{\xi})(\xi_j - \bar{\xi})}{\sum_i (\xi_i - \bar{\xi})^2} \tag{3.5}$$

where n is the number of trees, $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)^{\mathrm{T}}$ are the correlated random numbers, $\bar{\xi}$ is the mean of $\boldsymbol{\xi}$, and $w_{ij}(h)$ is a matrix of spatial weights. The spatial weights are defined as

$$w_{ij}(h) = \begin{cases} 1 & \text{if } i \neq j \wedge (h-1)\delta \leq d_{ij} < h\delta \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

where d_{ij} is the distance between the trees i and j, and δ is the lag distance increment.

The correlation was measured between zero and 30 metres in lag distance incre-

ments of 0.5 metres and is plotted as a correlogram in figure 3.8. The minimum and maximum correlation measured at a particular distance is represented by the blue area, while the theoretical correlation, i.e. the one given by formula (3.3), is represented by the red line. The maximum deviation between the correlation structure measured in these tests and the theoretical one was 0.09.

3.5 Performance and scalability

In order to analyse the performance gain and the scalability of the local Cholesky decomposition, two types of benchmark tests were conducted. The performance of the local Cholesky in comparison with the global Cholesky decomposition was tested by running exactly the same test case on both simulator implementations with a progressively increasing problem domain size. Furthermore, the parallelisation of the local Cholesky implementation was tested by running the same simulation on an increasing number of processors.

In all test cases the performance was analysed by observing the simulation wall clock time and peak memory usage. In the simulations an initial stand with a homogeneous spatial tree distribution and random tree marks was generated, and only a single time step involving growth was simulated. The Cholesky decomposition was performed in the growth phase to obtain the environmental effects.

3.5.1 Performance gain of the local Cholesky decomposition

The performance of the local Cholesky decomposition compared with the conventional global Cholesky decomposition was analysed by running exactly the same test case on both simulator implementations. The simulation was then repeated with a progressively increasing number of trees in order to identify the general trend in computation time and memory usage growth in both algorithms. The tree density was held constant at 1,000 trees per hectare in all tests, whereas the simulated area was widened proportionally as the number of trees increased. Because the implementation of the global Cholesky decomposition in previous versions of SPATE-HPC was not parallelised this test was performed using only a single processor.

The wall clock time of the local and the global decomposition is shown in fig-

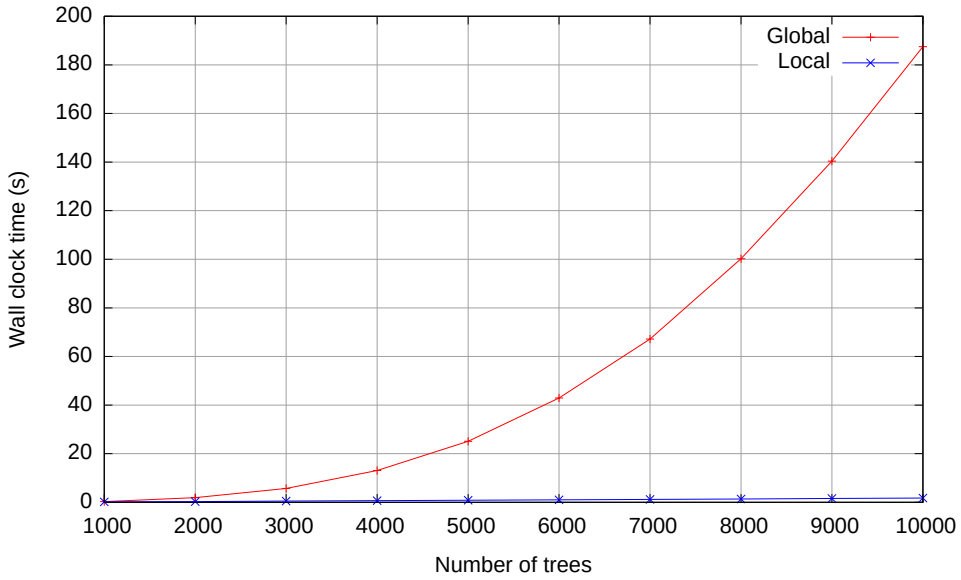


Figure 3.9: Simulation wall clock time using the global and the local Cholesky decomposition.

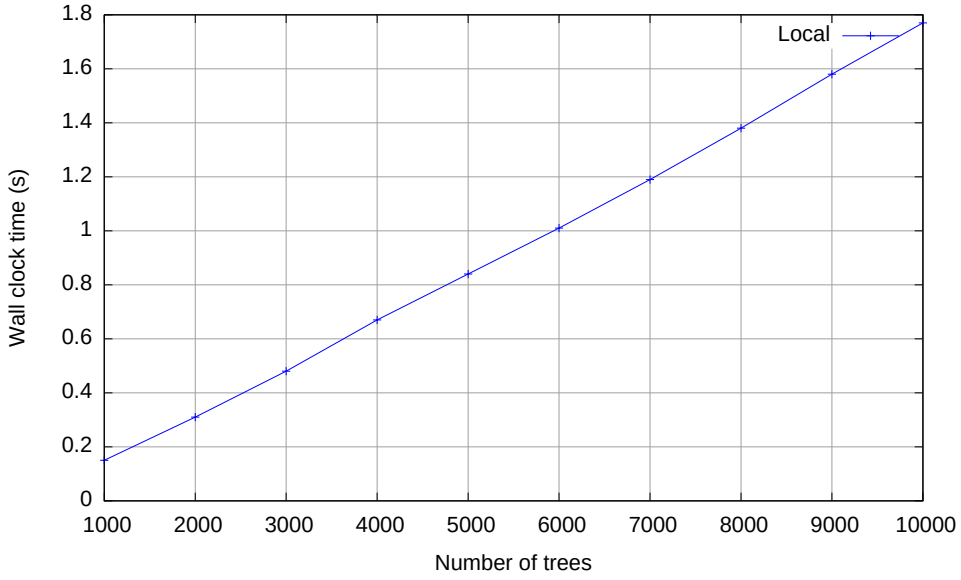


Figure 3.10: Simulation wall clock time using the local Cholesky decomposition.

ure 3.9. The figure exhibits a cubic growth for the global decomposition while the local decomposition is significantly faster. The general trend in the local decomposition wall clock time is not even discernible from this figure as the plotted line hovers just above zero.

Because the overall stand density is held constant between simulations, the average size of a covariance matrix in the local decomposition should also remain constant as well, with only the total number of individual decompositions increasing with the growing problem size. The local decomposition implementation should hence experience a linear computation time growth in this particular test case. Plotting the local decomposition alone in figure 3.10 indeed indicates a linear computation time growth.

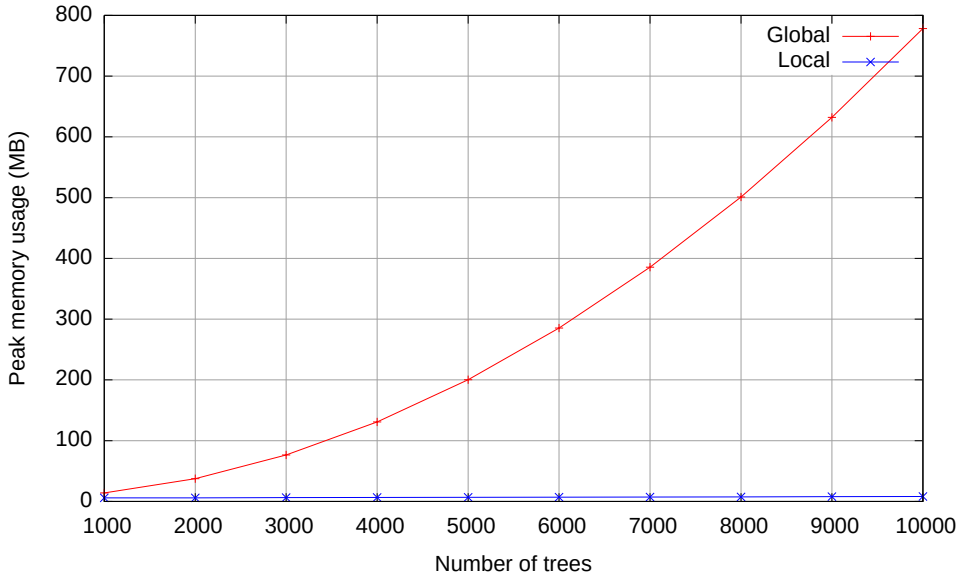


Figure 3.11: Peak memory usage using the global and the local Cholesky decomposition.

The memory usage of the local and the global decomposition is shown in figure 3.11. The quadratic memory usage growth of the global decomposition is clearly visible while the memory usage of the local decomposition is significantly smaller. Plotting the local decomposition alone in figure 3.12 indicates a linear growth, which is also to be expected.

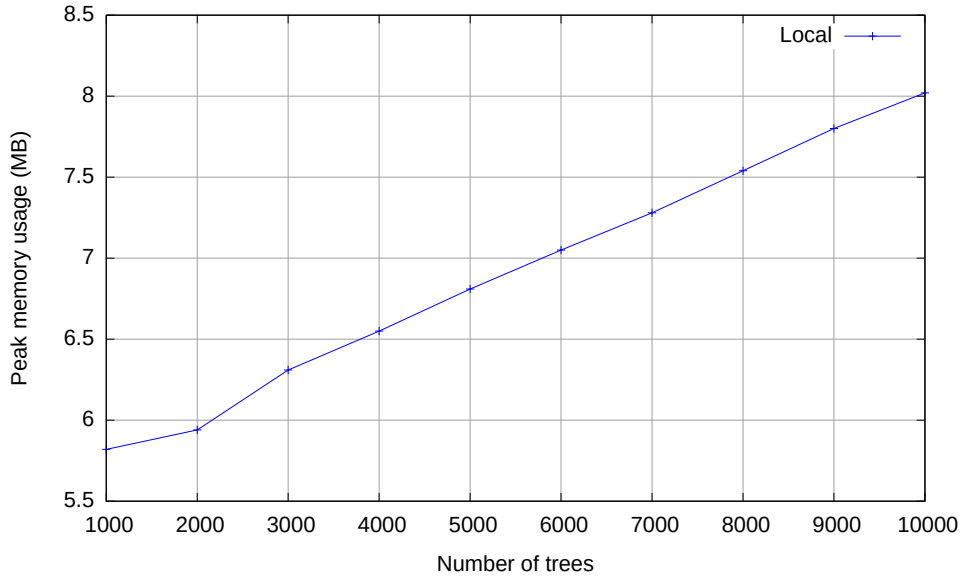


Figure 3.12: Peak memory usage using the local Cholesky decomposition.

3.5.2 Parallel scalability

The parallelisation of the local Cholesky decomposition algorithm was tested by running exactly the same simulation first on a single processor and then on an increasing number of processors. For this test, CSC's (IT Center for Science Ltd.) supercomputer *Louhi* was used. Louhi is a Cray XT4/XT5 hybrid system [17].

The simulated area was 1,600 ha and contained 6 million trees. The methodology for generating the correlated random numbers in this test was the same as when the global and the local Cholesky decompositions were compared; first an initial stand was generated and then a single growth phase with environmental effects was simulated. The elements of interest in this study are hence the decrease in wall clock time and memory usage with respect to the increase in number of processors. In an ideal parallelisation, twice the number of processors would entail half the wall clock time and half the memory usage per processor.

The total wall clock time of the simulation is shown in figure 3.13. As the graph shows, the computation time scaling is in fact slightly better than linear. This superlinear scaling phenomenon can be ascribed to better cache utilisation, as it is more likely that processors do not need to discard cached data and re-read data from memory when a processor's own problem size, and hence its memory consumption, becomes smaller. A general superlinear scaling in SPATE-HPC was also observed by Schöring [6].

The memory usage per processor is shown in figure 3.14. This graph, however, suggests a memory usage scaling that is slightly worse than linear. This behaviour is actually to be expected, as each processor runs its own instance of the simulator program and therefore carries a certain memory overhead. The overhead is roughly the same in all processors and its size depends on the simulated area and the input parameters.

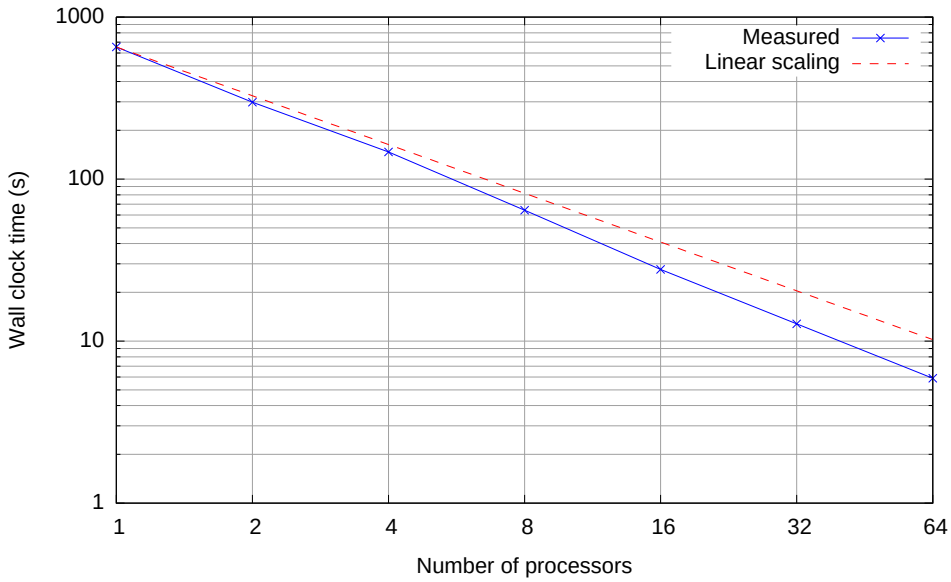


Figure 3.13: Simulation wall clock time when increasing the number of processors.

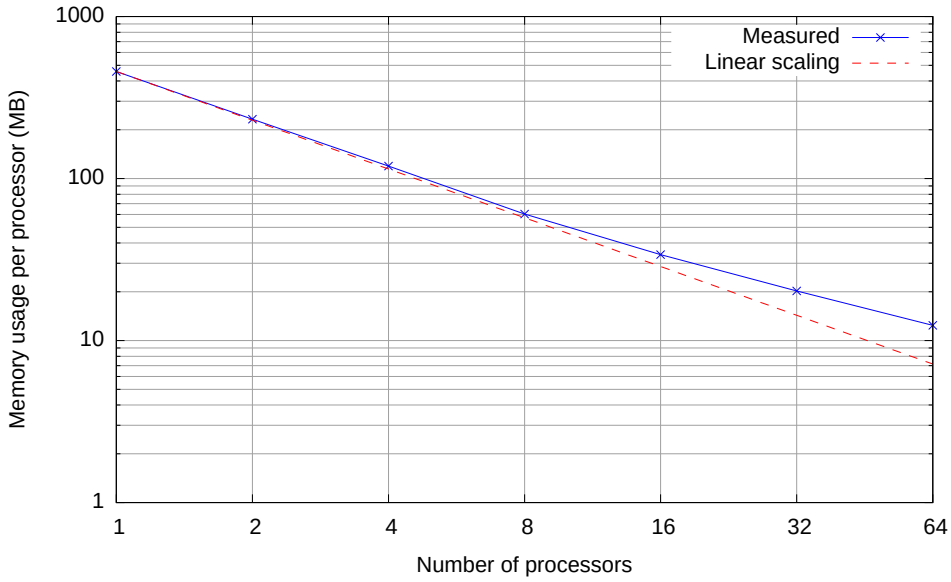


Figure 3.14: Memory usage per processor when increasing the number of processors.

4 TRANSFORMATION OF TREE CHARACTERISTICS

When new trees are generated for the initial stand and during the reproduction phase, the tree characteristics, or marks (diameter and height), of a tree are calculated according to formula (2.2). Due to the ε and ξ components of (2.2), y will be a normally distributed random variable, theoretically containing both negative and positive values. A normal distribution of initial tree marks may or may not be the preferred distribution, but having negative diameters and heights is definitely not desirable.

In previous versions of SPATE-HPC, the outcome of equation (2.2) was subjected to an exponential transformation, i.e. $\bar{y} = \exp(y)$. Hence, $\bar{y}$ was log-normally distributed. This eliminated the problem of negative mark values. However, a log-normal distribution of tree marks might not always be an accurate representation of a real tree population. To be able to simulate arbitrary tree stands, one should be able to generate tree marks having any kind of distribution.

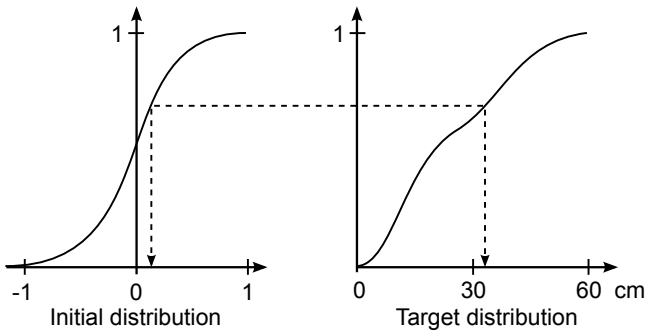


Figure 4.1: Transforming a value from one distribution to another via cumulative distributions.

This problem was solved by calculating the initial marks according to formula (2.2) and then transforming them to adhere to a target distribution. This is done by first cal-

culating the values of a particular mark type for all trees in the forest. A cumulative distribution is then built based on the normally distributed initial marks, which is used together with a target cumulative distribution to transform the marks. The transformation is done by mapping a function value of the initial cumulative distribution to the function value of the target cumulative distribution, as shown in figure 4.1. Formally, if $F_1(y), F_2(y) \in [0, 1]$ are cumulative distribution functions of the initial and the target distribution, respectively, then the transformed mark $\bar{y}$ is given by

$$\bar{y} = F_2^{-1}(F_1(y))$$

where $F_2^{-1}(y)$ is the inverse function of $F_2(y)$.

The process is applied first to the diameter mark and then to the height mark. There can be a separate target distribution for each mark, but since the height can be made a function of the diameter, the transformation of the height mark can be omitted.

4.1 Target distributions

Target distributions are specified in the XML input file for each mark and each forest type. Different forest types may hence have completely different distributions of marks in the initial stand and the reproduction phase. Four different ways of specifying target distributions were implemented: a normal distribution, a Weibull distribution, empirical data, and an empirical cumulative distribution.

Empirical data points are provided as an external file. The numerical values in the file, which may occur in any order, are read, and a target cumulative distribution is calculated accordingly. This method might be suitable if a large-enough amount of empirical data have been collected and the intention is to have the resulting initial tree stand adhering to the corresponding distribution.

If empirical data points are not available, a target cumulative distribution can also be generated from a theoretical normal or Weibull distribution. For a normal distribution, the mean and the standard deviation are given as parameters. For a Weibull distribution, the scale, shape and origin parameters are given.

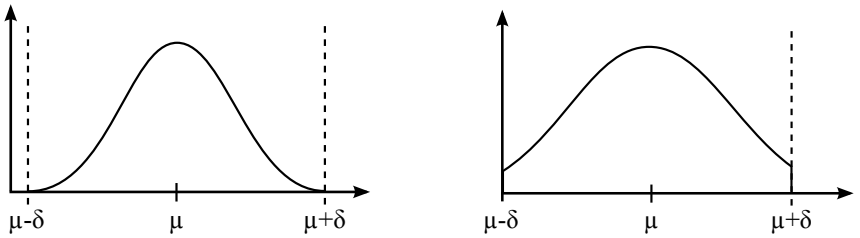
Finally, a completely arbitrary distribution may be provided by specifying the values of an empirical cumulative distribution provided as an external data file. The values in the file appear pairwise, so that the first value is the function parameter and the second value is the function value of the cumulative distribution function.

4.2 Normal cumulative distribution

If the user specifies a normal distribution as the target distribution, a cumulative distribution is calculated according to the given mean and standard deviation. Since a normal distribution has a support comprising the entire set of real values $(-\infty, +\infty)$, it must somehow be truncated in order to be implemented in a computer program. It was deemed feasible to restrict the support to an interval encompassing three standard deviations in each direction from the mean since, according to the empirical rule, three standard deviations account for approximately 99.7% of all values drawn from a normal distribution [18]. Moreover, because negative values cannot be allowed, the support is further truncated should the standard deviation and the mean be chosen so that the lower end of the support would otherwise be negative. The support $x \in [a, b]$ is calculated as

$$\begin{aligned} \delta &= \min\{3\sigma, \mu\} \\ a &= \mu - \delta \\ b &= \mu + \delta \end{aligned} \tag{4.1}$$

where μ is the mean and σ is the standard deviation. Figure 4.2a shows a normal distribution that encompasses three standard deviations in each direction, while figure 4.2b shows a normal distribution that is truncated not to include negative values.



(a) Slight truncation.

(b) Symmetrical truncation avoiding negative values.

Figure 4.2: Probability distribution of a truncated normal distribution.

According to its definition the cumulative distribution function (CDF) of a normal

distribution can be calculated as [19]

$$\Phi(x) = \int_{-\infty}^{x} \varphi(t)\,\mathrm{d}t = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)\,\mathrm{d}t. \tag{4.2}$$

The distribution should, however, be restricted to the reduced support. This can be done by truncating the distribution to the interval $[a, b]$ and subsequently scaling it, so that the total integral of the probability density function remains equal to one. The resulting CDF is

$$F(x) = \frac{\int_a^x \varphi(t)\,\mathrm{d}t}{\Phi(b) - \Phi(a)} = \frac{\Phi(x) - \Phi(a)}{\Phi(b) - \Phi(a)}. \tag{4.3}$$

Furthermore, the integral $\Phi(x)$ can be approximated by the error function using a Taylor series expansion [19],

$$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \tag{4.4}$$

where

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{n!(2n+1)}. \tag{4.5}$$

4.3 Weibull cumulative distribution

If the user specifies a Weibull distribution as the target distribution, a cumulative distribution is calculated according to the given scale, shape and origin parameters. The cumulative distribution function of the Weibull distribution [20] can be calculated as

$$F(x) = 1 - \exp\left[-\left(\frac{x-a}{\lambda}\right)^k\right] \tag{4.6}$$

where λ is the scale, k is the shape and a is the origin, also often referred to as the location.

The Weibull distribution has the support $x \in [a, \infty)$. However, in this implementation, the upper end of the support is approximated by a real number b, so that $F(b)$ is reasonably close to 1, i.e. $F(b) = 1 - \varepsilon$. Solving for b yields

$$b = \lambda \sqrt[k]{-\ln \varepsilon} + a \tag{4.7}$$

which can be used to find an approximate upper-end of the support.

4.4 Implementation

The program code for transforming tree marks comprises several classes: *MarkTransformation*, *MarkTransformer* and the classes implementing the interface *MarkDistribution*. *MarkTransformation* contains information about the target distribution, which is set up by a *MarkDistribution* object, while *MarkTransformer* performs the actual transformation of tree marks from the initial to the target distribution. Furthermore, *MarkTransformer* is used from within the *generateTreeMarks* method of the *InitialCompartmentGenerator* class when tree marks are generated for the initial stand and from the *calculateNewTreeMarks* method of the *Forest* class when tree marks are generated for new trees in the reproduction phase. The class diagram of the mark transformation classes is depicted in figure 4.3. In figure 4.4 the sequence diagram of how marks are transformed for the initial stand is presented.

4.4.1 MarkTransformation

Each *ForestType* instance contains two sets of *MarkTransformation* objects. There is one object for each mark and separate sets for the initial stand and the reproduction phase. When a forest type is loaded from the input parameter file and it contains a target distribution, the parameter loading is delegated to the *MarkTransformation* class, which, depending on the type of target distribution the user has chosen, instantiates a class of the *MarkDistribution* interface to generate the actual cumulative distribution.

4.4.2 MarkDistribution

Classes that implement *MarkDistribution* are *MarkNormalDistribution*, *MarkWeibullDistribution*, *MarkECDFDistribution* and *MarkDataDistribution* for a normal distribution, a Weibull distribution, an empirical cumulative distribution and a cumulative distribution from empirical data, respectively. If normal distribution parameters are given, *MarkNormalDistribution* will be used to compute a cumulative distribution according to formulae (4.3) and (4.4). If Weibull distribution parameters are given, *MarkWeibullDistribution* will be used to compute a cumulative distribution according

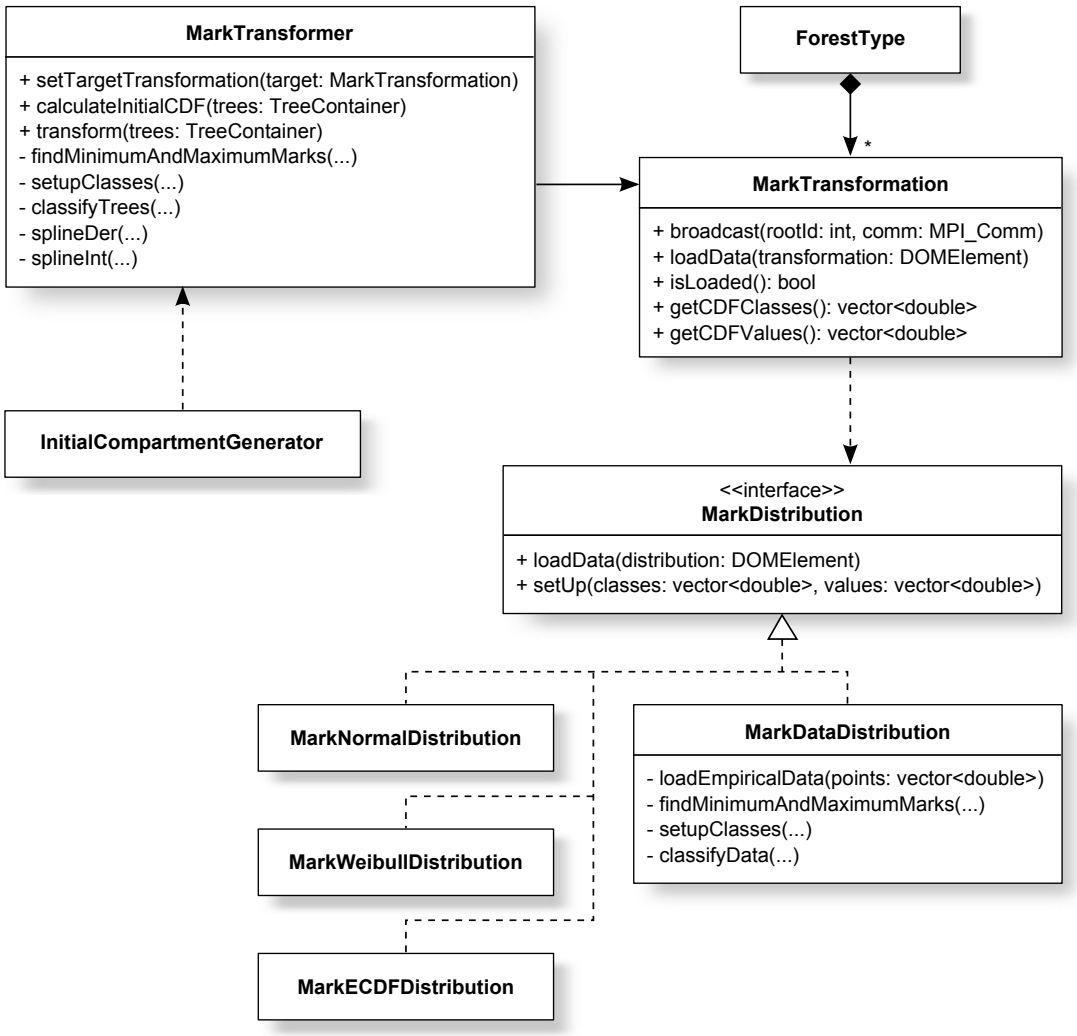


Figure 4.3: Class diagram of mark transformation classes.

to formulae (4.6) and (4.7). If a filename of an empirical cumulative distribution is given, *MarkECDFDistribution* will read the file and set up a cumulative distribution with the exact values from the contents of the file. Lastly, if a filename of an empirical data file is given, *MarkDataDistribution* will read the file, count the frequencies of values, and compute a cumulative distribution accordingly.

4.4.3 MarkTransformer

An object of class *MarkTransformer* is given a mark ID and a *MarkTransformation* object, indicating which mark to transform and to which target distribution. Firstly, the *calculateInitialCDF* method is called. The method begins by finding the smallest

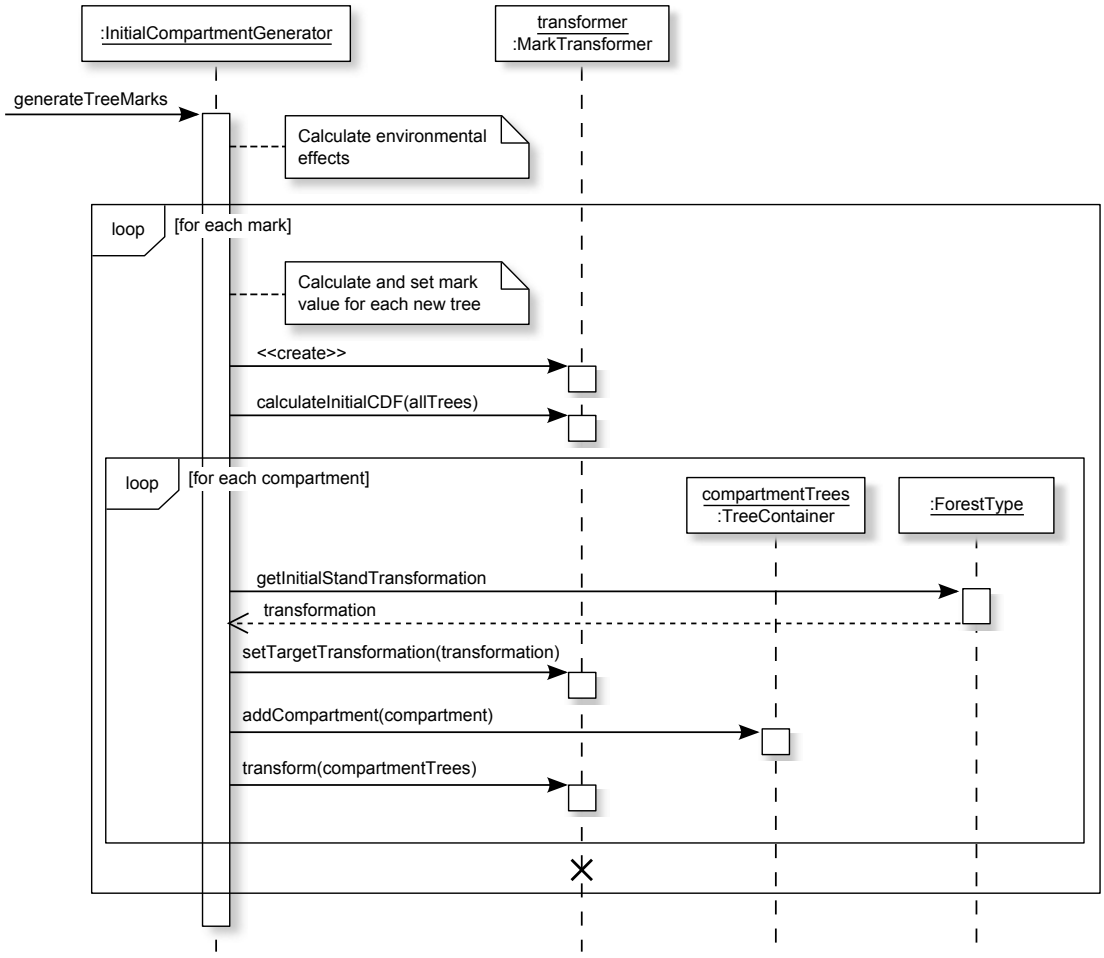


Figure 4.4: A sequence diagram showing how *InitialCompartmentGenerator* transforms tree marks for the initial stand.

and largest mark values. This interval is then divided into 100 size classes, which are used to classify the trees; the number of trees that fall into a certain class are counted to produce a frequency distribution. The frequency distribution is then used to build a cumulative distribution that can be used for the transformation process.

Finally, to perform the actual transformation, the *transform* method is called. The new tree marks are calculated by translating from the initial to the target cumulative distribution using cubic spline interpolation [21].

5 TRUNK AND ASSORTMENT VOLUME ESTIMATION

A significant benefit from simulating forest growth is the prospect of obtaining statistics on timber volumes. Both a tree's total volume as well as volumes of different tree segments may be useful for assessing gross wood resources and yield quantities of different types of wood raw materials.

Because the simulator does not maintain information about the entire shape of a tree, merely its total height and diameter at breast height, its volume is something that must be estimated from these two values with a mathematical model. Models for calculating the volume of common Finnish tree species are presented in Tapion Taskukirja [22] and by Laasasenaho [23].

Earlier versions of SPATE-HPC used a model from Tapion Taskukirja. The model, however, could not be used to estimate the volume of arbitrary trunk segments but merely the tree's volume in its entirety. The model was therefore replaced by Laasasenaho's taper curve formulae. Given a tree's diameter and height, Laasasenaho's formulae estimate the tree's diameter at an arbitrary height position. The formula implemented in SPATE-HPC is formula 33.1 by Laasasenaho [23]:

$$\frac{d_l}{d_{.2h}} = b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^5 + b_5 x^8 + b_6 x^{13} + b_7 x^{21} + b_8 x^{34}. \tag{5.1}$$

Here, d_l is the diameter at height l from the ground, $d_{.2h}$ is the diameter at 20% of the tree's height, $x = l/h$, and $b_1, \ldots, b_8$ are species-specific coefficients. Although the exact value of $d_{.2h}$ for a tree is not known within the simulation, it can be estimated. Let $f_b(x)$ be equal to formula (5.1). By setting $l = 1.3$, an estimate $\hat{d}_{.2h}$ for $d_{.2h}$ can be obtained as

$$\hat{d}_{.2h} = \frac{d_{1.3}}{f_b(1.3/h)} \tag{5.2}$$

where $d_{1.3}$ is the DBH and h is the tree's height [23]. Figure 5.1 shows the tapering

diameter curves of three sample trees calculated with formula (5.1).

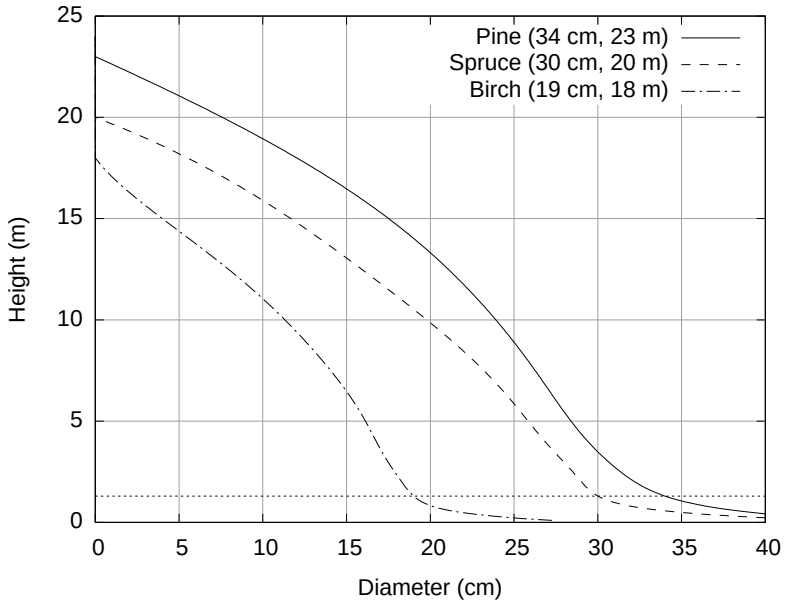


Figure 5.1: The tapering diameters of a sample pine, spruce and birch tree with the DBH's 34 cm, 30 cm and 19 cm, and the heights 23 m, 20 m and 18 m, respectively.

When a tree's diameter at arbitrary height positions can be estimated, the tree's volume is readily calculated by integrating; integrating from ground to top yields the total volume, whereas integrating a particular segment yields the volume of that segment, i.e.:

$$V_{a,b} = \pi \int_a^b \left(\frac{d_l}{2}\right)^2 \mathrm{d}l. \tag{5.3}$$

As Laasasenaho points out, the formulae are not accurate for height positions very close to the ground, but since trees in practice are cut at a certain height above the ground to leave a stump, this poses no problem [23]. When calculating a tree's entire volume one can simply start at the height of a reasonable stump and integrate towards the top of the tree. The stump height of a tree is calculated with the following linear model:

$$h_s = (b_1 d_{1.3} + b_2 h)/100 \tag{5.4}$$

where h_s is the stump height in metres and b_1 and b_2 are species-specific coefficients.

Unfortunately, formula (5.1) has the disadvantage that it cannot be used for small trees in this simulator. Because the simulator does not maintain information about the

diameter at 20% of the height, and the only two input parameters are the DBH and the total height, the output becomes meaningless for trees shorter than 1.3 metres. In practice, the formula is also unreliable for trees slightly taller than 1.3 metres, since the DBH would have to be measured with very high precision very close to the top. Nonetheless, SPATE-HPC does support trees shorter than 1.3 metres that still have a positive DBH mark value, as this supposed discrepancy does not necessarily pose a problem in the growth models. For very small trees, the DBH can simply be interpreted as the diameter at any reasonable height position.

In order to calculate the volume of small trees, however, alternative procedures must be introduced. In SPATE-HPC, this is solved by switching to the simple cylinder model

$$V = \pi(d_{1.3}/2)^2 c \tag{5.5}$$

where c is a species-specific weight constant, typically around 0.6 [22]. A tree is considered small in SPATE-HPC if its DBH is smaller than 3 centimetres or its height shorter than 4 metres. The cylinder model might not be as accurate as the taper curve model, but since trees of this size cannot be used for logging [22], it is considered sufficient for this particular purpose.

5.1 Assortment volumes

The term *assortment* is used to describe a particular type of wood raw material that is obtained by cutting pieces off a felled tree and is later processed into some kind of wood produce [24]. Examples of such assortments are sawlog, pulpwood and energy wood. Wood of a certain assortment is often cut into logs of a specific length with a diameter of a minimum width. A key requirement of SPATE-HPC is to be able to produce statistics on different assortment volumes after a simulation has been performed [2].

The algorithm implemented in SPATE-HPC to calculate assortment volumes is fairly straightforward. As its criteria an assortment may have a log length and a minimum diameter. Logs of that length are then cut off the tree if the trunk diameters at both ends of the log are wide enough to accommodate the minimum-diameter criterion. Different assortments may have different size criteria and, additionally, assortments of different species may have different size criteria as well.

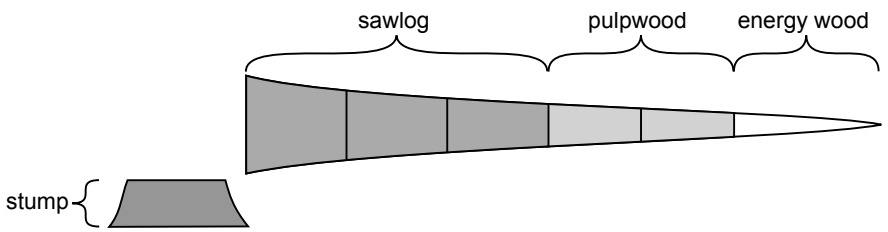


Figure 5.2: A sample tree trunk cut into the assortments sawlog, pulpwood and energy wood.

The order in which assortments are specified is relevant. The first assortment takes precedence over the second one, the second one over the third one and so forth. The cutting process starts at the stump, attempting to produce logs of the first assortment. This continues until the tree's diameter no longer fulfils the minimum diameter criterion, after which the algorithm switches to the next assortment and proceeds by likewise cutting logs of that assortment. The process continues until the remaining part of the tree cannot be used for any assortment. At this point, if a last assortment without criteria is specified, the final part of the tree is used for that assortment. Figure 5.2 shows an example of how a tree trunk can be cut into different assortments.

5.2 Implementation

Several classes are involved in the process of calculating trunk volumes. The class *DiameterModel* implements formula (5.1) and (5.2) for estimating the diameter at arbitrary height positions, the class *StumpModel* implements formula (5.4) for calculating the stump height, the class *VolumeModel* integrates over a given trunk segment using numerical integration in order to obtain the segment's volume, and the class *AssortmentVolumeModel* implements the algorithm described in section 5.1 for calculating the assortment volumes of trees. Additionally, the classes *AssortmentContainer* and *Assortment* are used to maintain a list of assortments and their respective size criteria. Figure 5.3 shows a class diagram of the classes and their mutual dependencies.

The process of calculating assortment volumes of an individual tree is illustrated in the activity diagram in figure 5.4. The procedure is initiated with a call to the method *calculateForTree* of the class *AssortmentVolumeModel*. The method then uses the *estimateDiameterAt20PercentOfHeight* of the *DiameterModel* class to estimate the

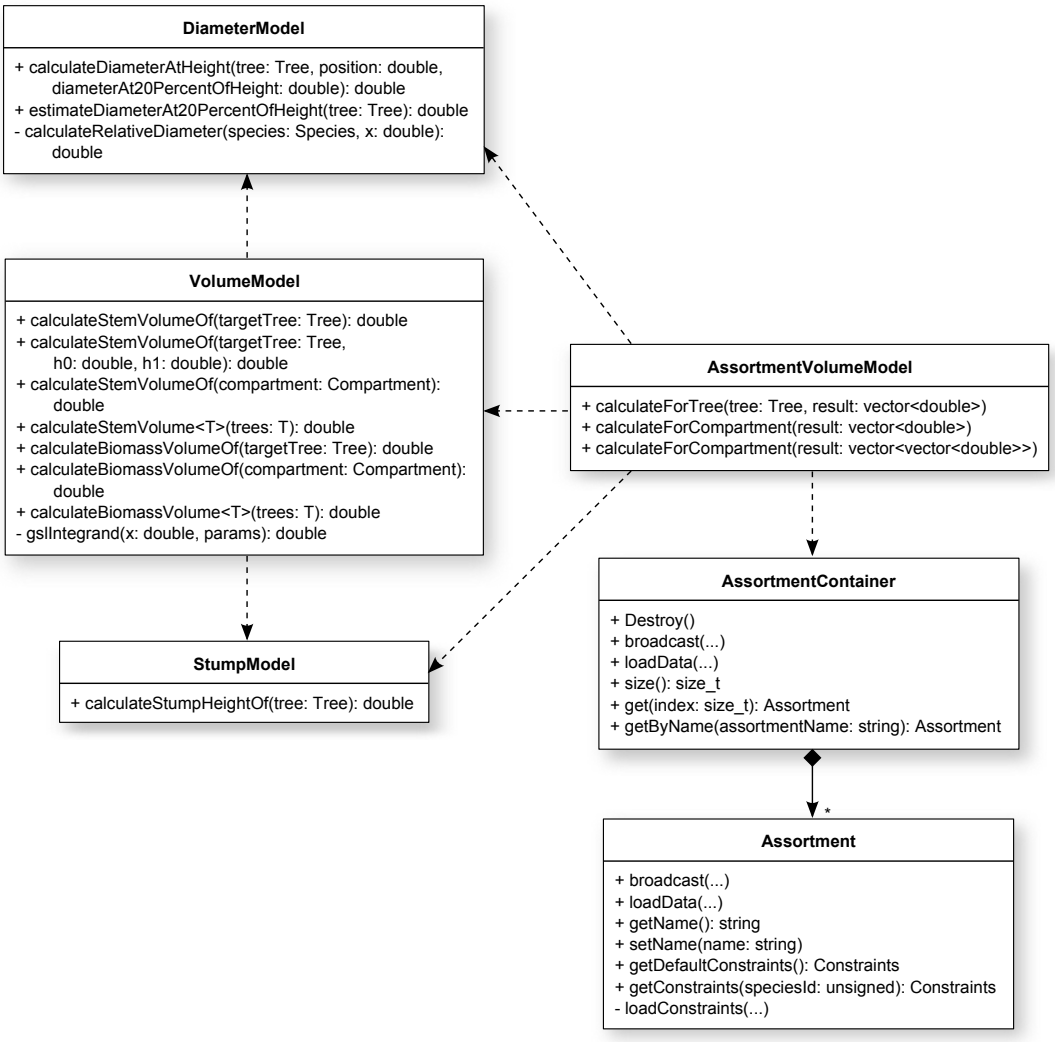


Figure 5.3: A class diagram showing the relationship between classes used to calculate the trunk and assortment volumes of trees.

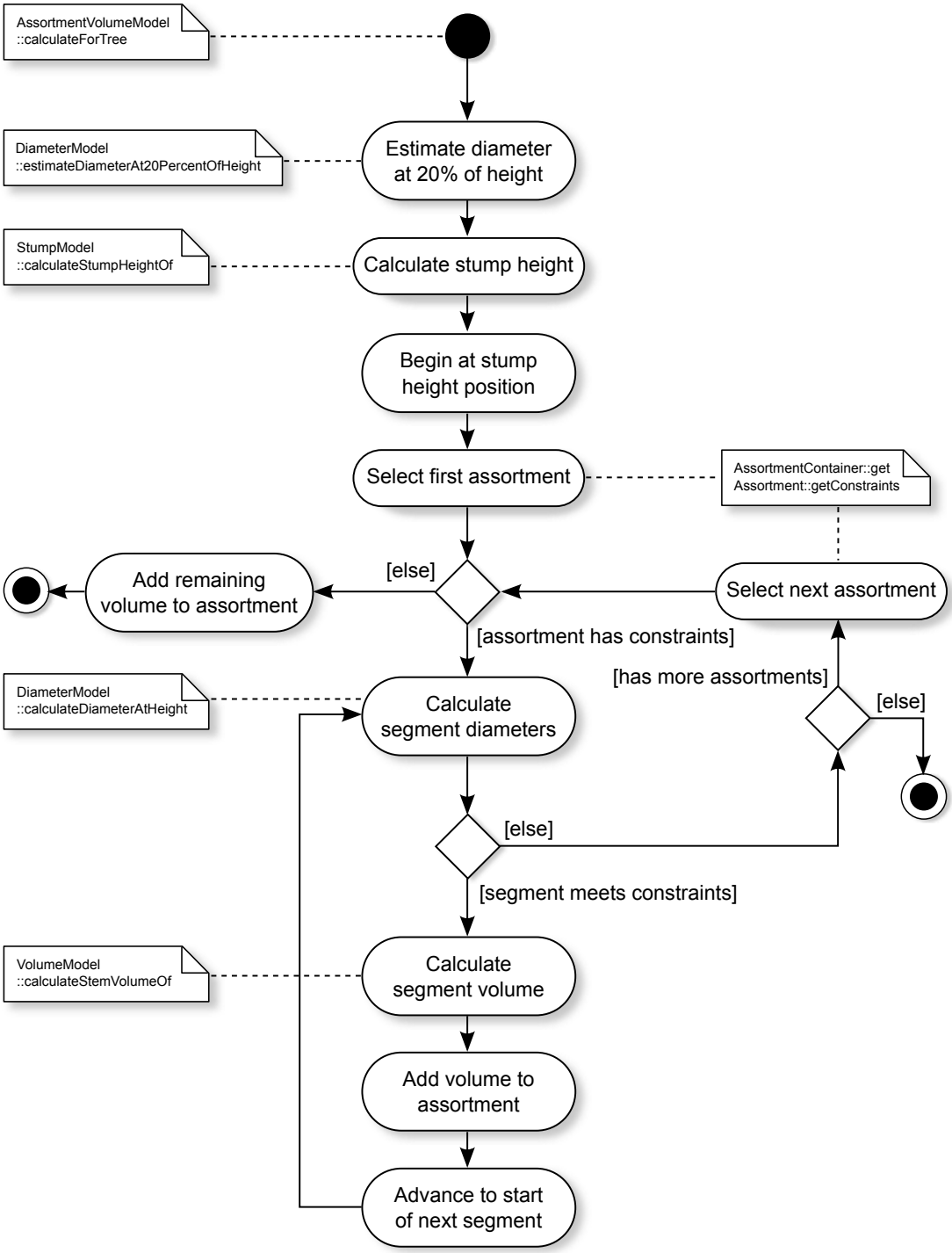


Figure 5.4: An activity diagram showing the process of calculating assortment volumes of a tree.

diameter at 20% of the height and the *StumpModel* class to calculate the stump height. The next step is to iterate through the list of assortments obtained from the *AssortmentContainer* class and add log volumes to the current assortment if possible. The implementation uses the *calculateDiameterAtHeight* method of the *DiameterModel* class to calculate the diameter at both ends of the log and the *VolumeModel* class to calculate the log's volume.

6 WEIBULL THINNING

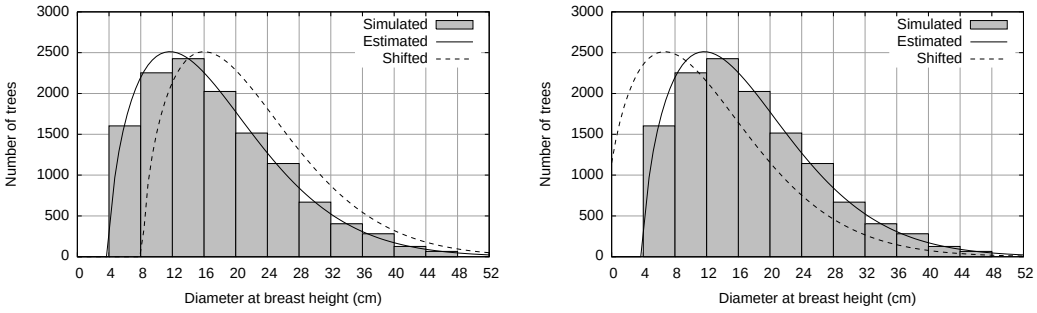
Silvicultural treatments are an essential functionality of SPATE-HPC. Silvicultural treatments include both thinning practices that selectively remove individual trees in order to improve the growth of the remaining trees and felling in order to harvest wood raw material. Both categories of silvicultural treatments are implemented in SPATE-HPC as an array of different management methods. The management methods in SPATE-HPC are (a) dimension cutting and (b) energy wood cutting, which deterministically cut trees larger or smaller than a specific diameter limit, respectively; (c) low thinning and (d) selective thinning, which stochastically cut small or large trees, respectively; and (e) clearcutting, which cuts down most or all of the trees.

The implementation of low and selective thinning, which in SPATE-HPC are collectively referred to as Weibull thinning, have undergone significant redesign in recent versions of the simulator. This chapter describes how low and selective thinning have been implemented in the current version of SPATE-HPC.

6.1 Size classes

In low and selective thinning, thinning is applied collectively to trees in different size classes. Trees are categorised based on their DBH into size classes of a specific user-defined width, with the default class width being 4 cm. The diameter distribution of the trees can then be analysed by counting the number of trees in each class and drawing a histogram.

Furthermore, a Weibull distribution [20] is estimated so that it resembles the actual distribution of the simulated tree population. The curve that represents the Weibull distribution is subsequently shifted either left or right. It is shifted to the left when selective thinning is applied and to the right when low thinning is applied. The shifted Weibull curve is likely to intersect the histogram bars either at the left or right hand



(a) Low thinning

(b) Selective thinning

Figure 6.1: The simulated tree distribution is represented by the histogram and the estimated distribution by the solid curve. The dashed curve is the shifted curve.

side of the diameter distribution histogram. This is shown in figure 6.1.

Thinning is applied to the size classes whose histogram bars the shifted Weibull curve intersects. The objective is to thin trees in such a way that the amount cut in each size class represents the part of the histogram bar that protrudes above the shifted curve. This bar-to-curve ratio constitutes the *thinning rate*, which is a fundamental element of the thinning procedure. The thinning rate h_k of a size class k is calculated as

$$h_k = \begin{cases} 1 - w_k/n_k & \text{if } n_k > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where w_k is the value of the shifted Weibull curve at the centre diameter of class k, and n_k is the number of trees in class k. If $h_k < 0$, no thinning is applied to that particular class.

6.2 Low thinning

The basis for the low thinning method is the *size class dependent thinning rule* [3]. In low thinning trees in each size class are thinned uniformly. All trees in a size class share the same thinning probability and the likelihood of a tree being cut does not depend on the tree's location or other surrounding trees.

When low thinning is applied, the estimated Weibull distribution curve is shifted to the right, as shown in figure 6.1a. Low thinning starts with the smallest size class and proceeds towards larger ones. The last size class to which thinning is applied is

the size class where the two Weibull curves intersect. The probability of an individual tree being cut is equal to the thinning rate of the tree's size class. For each tree in a class, a uniform random number $u \sim U(0, 1)$ is drawn. If $h_k \geq u$, the tree is marked for removal.

6.3 Selective thinning

Selective thinning is based on the *stochastic thinning strategy with fixed thinning probabilities* method [3]. In selective thinning the aim is to cut large trees while simultaneously attempting to even out the overall stand density. The thinning probability of an individual tree in selective thinning therefore depends on the density of the local neighbourhood; trees in denser neighbourhoods have a higher thinning probability than those in sparser neighbourhoods.

Contrary to low thinning, the estimated Weibull curve is shifted to the left in selective thinning. This is shown in figure 6.1b. Selective thinning starts with the largest size class and proceeds towards smaller ones. The last size class to which thinning is applied is the size class where the two Weibull curves intersect.

For a given size class, selective thinning begins by assigning individual thinning probabilities to each tree. The thinning probability ω of a tree is calculated using the logistic model

$$\omega = \frac{1}{1 + \exp\big(-\beta_0 - \beta_1 \ln(\lambda_U/\lambda_A)\big)} \tag{6.2}$$

where β_0 and β_1 are parameters chosen by the user, λ_U is the local density around the target tree, and λ_A is the overall stand density [3]. After all trees in the class have been assigned thinning probabilities, the trees are sorted in descending order according to thinning probability.

The next step is to draw a Poisson random number p for the number of trees to cut in the class. The mean of p is the thinning rate multiplied by the number of trees in the class, i.e. $p \sim \mathrm{Pois}(h_k n_k)$. The algorithm then iterates p times. At each iteration step the algorithm starts with the first tree—the one with the highest thinning probability—and proceeds towards trees with lower thinning probabilities. Each uncut tree is tested by drawing a uniform random number $u \sim U(0, 1)$. If $\omega \geq u$ the tree is cut. Otherwise the procedure is repeated for the following tree until a tree has successfully been cut or the algorithm has failed to cut a single tree. When a tree has been cut or every

tree has been tested, the algorithm continues with the next iteration step and restarts from the first tree in the list, repeating the whole routine. After p iteration steps have been performed, a number of trees between zero and p have been cut and the thinning procedure is done.

6.4 Finding a good curve offset

A problem in low and selective thinning is to find a good offset for the shifted Weibull curve. In reality, the offset is an input variable to a stochastic process of which the resulting removal volume is an output value and the actual trees cut constitute a side effect. The offset, however, is usually of little concern to the user, who is likely more interested in specifying an actual volume to be removed. The user should therefore be able to give a target volume so that after thinning has been carried out the total removal volume would amount to approximately this target volume. Consequently, the offset has to be estimated in order to achieve the desired removal volume.

One conceivable way of estimating the offset would be to analyse the area of the curves or, more precisely, the area difference between the two curves where they overlap. The shifted curve could be offset so that the overlapping area corresponds to the target removal volume with respect to the total stand volume. It turns out, however, that this is a rather crude way of estimating the offset. Since trees with smaller diameters have a much smaller volume than trees with larger diameters, the diameter size distribution curves do not represent the volume distribution accurately; an area subsection of the right part of the diameter distribution curve will usually amount to a significantly larger volume than an equally-sized subsection of the left part of the curve. This is how the offset was estimated in previous versions of SPATE-HPC.

6.4.1 Iterative approach

A perhaps more suitable approach would be to find the target volume through an iterative numerical root-finding algorithm. Although the process is stochastic and hence non-deterministic, that is, different output values can be obtained with the same input, observations on its general behaviour can be made. The offset of the shifted curve directly affects the number of histogram bars the curve intersects and also the thinning rate of the corresponding size classes. A larger offset means more bars and higher thinning rates and, as a consequence, a larger removal volume. Furthermore, the extrema

of the process output—i.e. the smallest and the largest possible removal volumes—are found as the offset values for which no or all histogram bars are intersected by the shifted curve. If the target removal volume is achievable, it is must lie within the range bounded by these extrema. By testing values within this interval and successively narrowing it down, the target volume within a certain tolerance can be reached.

6.5 Linearising the tail of the Weibull curve

Since the support of the Weibull probability distribution function has no upper bound, the function's value approaches zero asymptotically. This might cause a problem in some cases, most notably in selective thinning if the tail of the Weibull curve is incongruous with the simulated data, which may occur if the Weibull distribution happens to be an exceptionally bad representation of the simulated data. If the histogram bars of the largest size classes are shorter than the height of the shifted Weibull curve in selective thinning, then, according to the algorithm, the thinning procedure will not cut trees in the largest size classes but only in smaller size classes. Because selective thinning is generally used to fell trees primarily in the largest size classes [3], this might not be the intended outcome. This is shown by the histogram and solid-line Weibull curve in figure 6.2. The Weibull curve in the figure is a particularly poor representation of the simulated data.

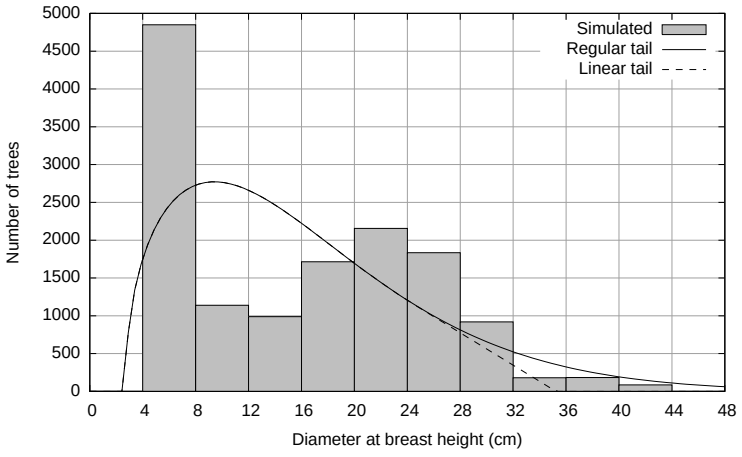


Figure 6.2: A graph showing a histogram of simulated data and a shifted Weibull curve (solid line). The Weibull curve's tail can be replaced with a linear one (dashed line).

A solution to this problem is to artificially transition the Weibull curve into a linear

function at its slope, thus forcing the curve to zero prematurely. The dashed line in figure 6.2 illustrates this. The linear tail starts when the ratio between the gradients of two consecutive size classes, both with negative slopes, is greater than a predefined constant. Formally, if $a_k/a_{k+1} > \alpha$, where a_k is the gradient at the centre of size class k and α is the predefined constant, the linear tail will start at the centre of class k and have the gradient a_k, while the rest of the Weibull curve is discarded.

6.6 Implementation

The different management methods in SPATE-HPC are implemented in an object-oriented fashion so that each management method is implemented as a separate class and management methods that share common concepts are subclasses of a common superclass. The abstract class *Management* is the superclass of all management classes and implements functionality that is common to all management methods. *SizeClassThinning* contains functionality for size-class-based management, *WeibullThinning* contains functionality related to Weibull thinning, and the classes *LowThinning* and *SelectiveThinning* implement the low and selective thinning procedures. Furthermore, the classes *WeibullFunction* and *LinearTailWeibullFunction* are auxiliary classes that are used to represent a Weibull curve and a Weibull curve with a linearised tail, respectively. Figure 6.3 shows a class diagram of the management classes.

6.6.1 WeibullThinning

The class *WeibullThinning* contains program code that is common to both low and selective thinning. The thinning procedure is initiated with a call to the *apply* method. The program then estimates a Weibull distribution for the simulated data and guesses an initial shift for the Weibull curve. The actual thinning procedure is performed by calling the method *applyIteration*, which is implemented in the *LowThinning* and *SelectiveThinning* classes. After the appropriate trees have been marked for cutting, the total removal volume is calculated. If the thinning volume falls within the range given by the target volume and a specified tolerance percentage, the thinning result, i.e. the trees marked for cutting, is accepted. Otherwise the thinning result is discarded and the program tries again with a different offset for the shifted Weibull curve, which is obtained using the *bisection method*.

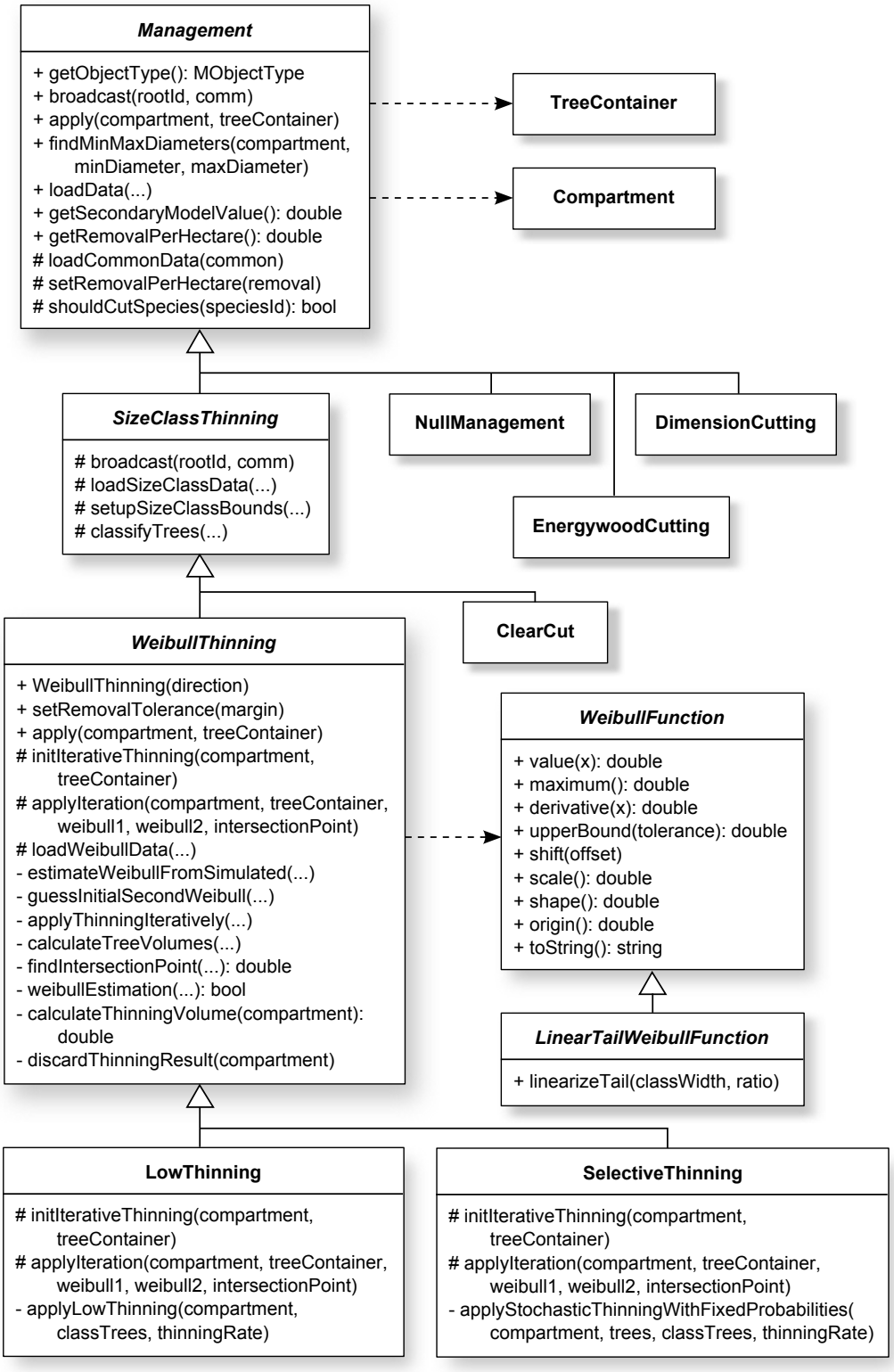


Figure 6.3: Diagram of the classes that implement management methods.

The bisection method [21] is a root-finding algorithm that operates by halving the interval where the root is known to lie into two usually equally-sized subintervals. Suppose, for the sake of simplicity, that $f(x)$ is a monotonically increasing deterministic function whose root lies within the interval $[a, b]$. If an arbitrary point $c \in [a, b]$ is chosen and $f(c) < 0$, the root must lie within $[c, b]$. Otherwise, if $f(c) > 0$, it lies within $[a, c]$. The value of c is usually the midpoint between a and b, and the procedure can be repeated until an x value of good-enough precision is attained. If the root exists, it will be found within $\lceil \log_2 \left((b - a)/\varepsilon \right) \rceil$ iterations, where ε is the tolerance for the root-finding algorithm [21].

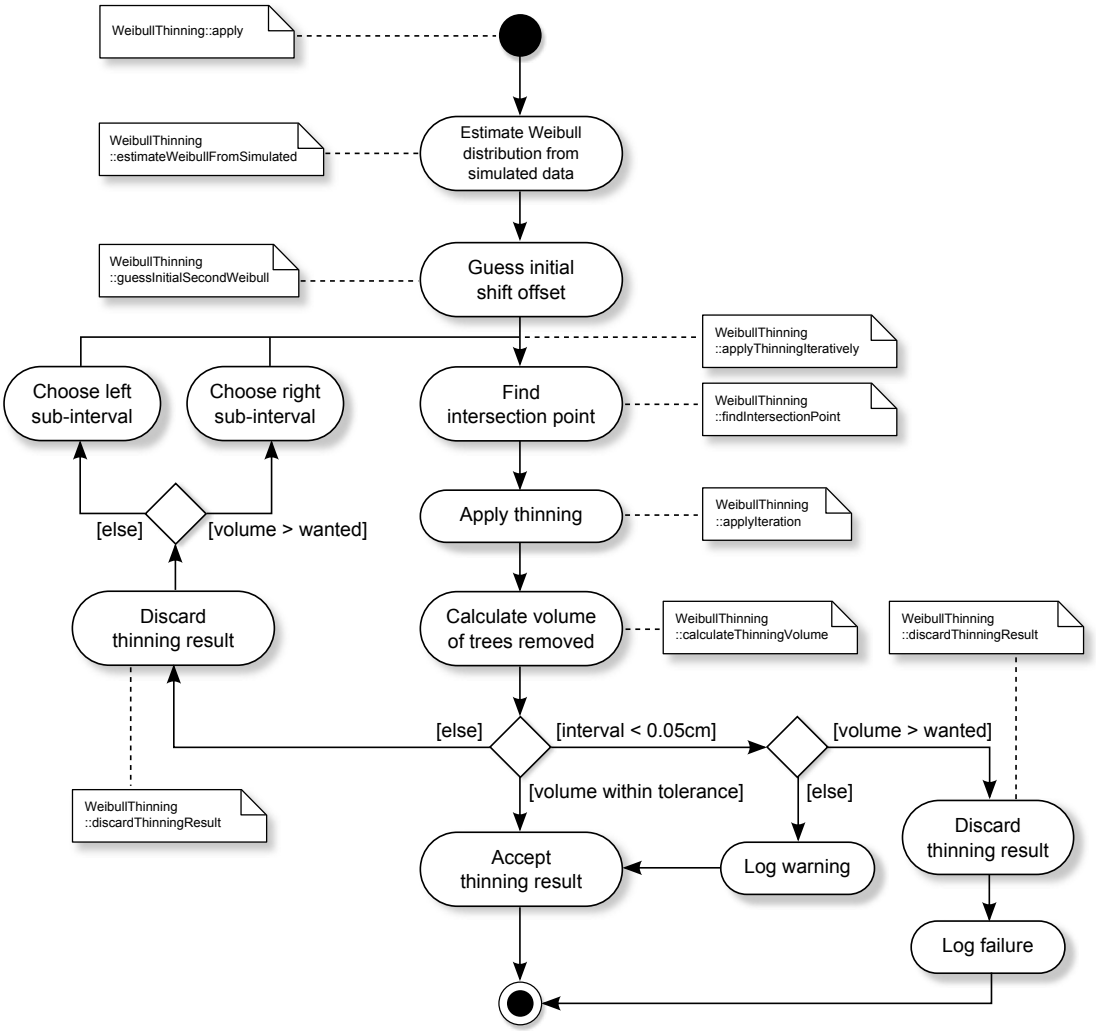


Figure 6.4: Activity diagram showing how the Weibull thinning procedure is performed within the *WeibullThinning* class.

Since the thinning process is non-deterministic, the output value might vary significantly with only slight changes in input value. Such behaviour may cause other numerical root-finding algorithms that also utilise the function's gradient to oscillate violently. Although the bisection method converges rather slowly compared with other numerical root-finding algorithms, such as the secant method and the false position method [21], it was deemed suitable for this purpose because of its robustness and immunity to oscillation.

Due to the thinning process' non-deterministic nature, it is possible that the next chosen subinterval is not the optimal one. It is even possible that the algorithm locks itself into an interval where the root cannot be found. A mechanism was therefore implemented to stop the iteration procedure if it becomes apparent that it will not terminate successfully. In general, the current subinterval will be less than 0.05 cm after approximately eight iterations. If this happens, and the target volume has not been reached, the procedure stops. If the resulting volume is smaller than the target volume, it is still accepted but with a warning message issued to the log file. A larger volume than the target volume, however, is not accepted. An activity diagram depicting the iterative Weibull thinning procedure is shown in figure 6.4.

6.6.2 LowThinning and SelectiveThinning

The *LowThinning* and *SelectiveThinning* classes extend the *WeibullThinning* class by implementing the method *applyIteration*. The method implements the low thinning procedure described in section 6.2 in the *LowThinning* class and the selective thinning procedure described in section 6.3 in the *SelectiveThinning* class, and it is called from within a Weibull thinning iteration step. The method traverses the trees according to size class and marks them for removal according to the size class' thinning rate, the random numbers drawn and, in case of selective thinning, the trees' individual thinning probabilities.

7 CASE STUDY

In order to test the implementation of tree characteristics transformation, assortment volume estimation and the Weibull thinning methods, a case study was conducted. The case study examined the functionality and synergy of the implemented methods when used together in a simulation of a real-world-like scenario. The case study consisted of two separate simulation set-ups, one with a low thinning management scheme and one with a selective thinning management scheme. Both simulations simulated the same forest area and started from the same initial stand.

7.1 Set-up

The simulated forest area was 59.6 ha and comprised 38 compartments of varying size. The simulated tree species were Scots pine, Norway spruce, birch and other broad-leaved trees. Most of the compartments were pine-dominant, while some were spruce-dominant or mixed. The forest area is shown in figure 7.1.

The initial stand was generated to have an average age of 40 years and a tree density of 2,000 trees per hectare, and the size structure of the trees was set up to imitate that of an uneven-aged forest. The target diameter distribution of the initial stand is shown in figure 7.2. The simulations were run with iteration steps of one year that included a growth, a mortality and a reproduction phase.

The low thinning scheme and the selective thinning scheme were set up to yield approximately the same removal volume. In the low thinning scheme four harvests were carried out. The first one was an energy wood cutting, which was followed by two low thinnings, the second one cutting twice as much as the first one. The fourth harvest was a clear cut. The selective thinning scheme was a nearly periodic thinning schedule in which the same selective thinning procedure was carried out repeatedly.

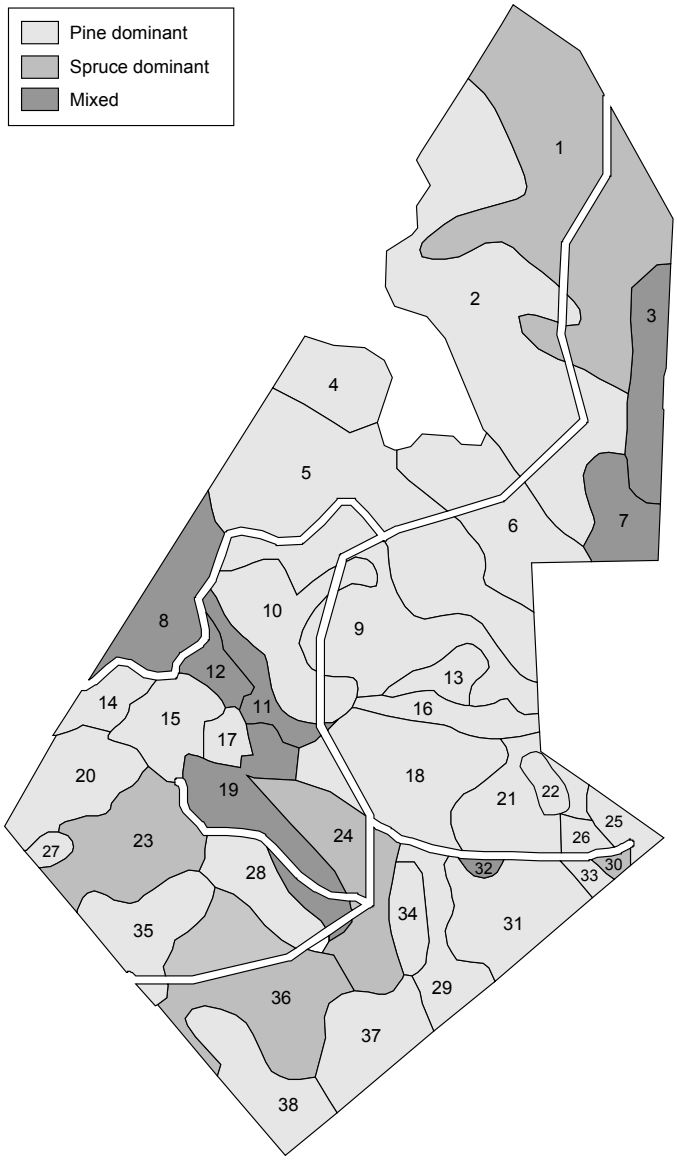


Figure 7.1: The 59.6 ha forest area comprising 38 compartments used in the case study simulations.

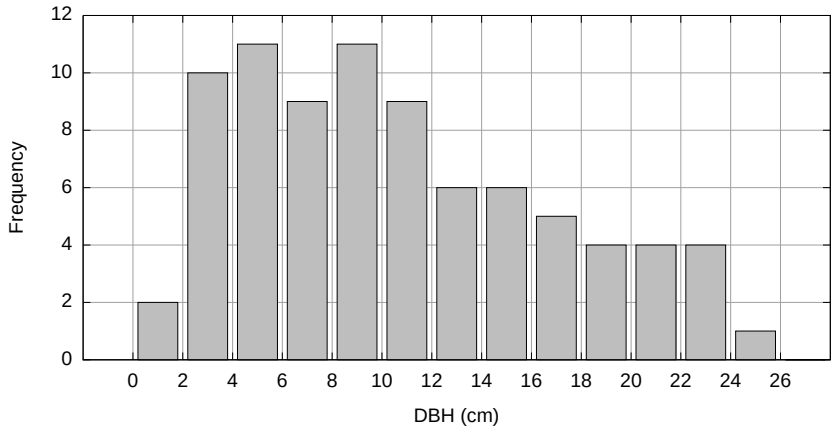


Figure 7.2: Target distribution of the diameter transformation for the initial stand.

The management methods in both schemes were set to be triggered when a compartment's growing stock or its total basal area reached a specified limit, and the thinning methods were set to cut trees in order to produce approximately the target removal volumes. Table 7.1 shows the management methods and their respective rules.

Table 7.1: The low and selective thinning schemes

Management	Criterion	Removal
Low thinning scheme		
1. Energy wood cutting	Basal area ≥ 20 m^2/ha	
2. Low thinning	Growing stock ≥ 200 m^3/ha	50 m^3/ha
3. Low thinning	Growing stock ≥ 250 m^3/ha	100 m^3/ha
4. Clear cut	Growing stock ≥ 250 m^3/ha	
Selective thinning scheme		
1–4. Selective thinning	Growing stock ≥ 250 m^3/ha	100 m^3/ha

The low thinning scheme was simulated for 70 years, while the selective thinning scheme was simulated for 110 years. The low thinning simulation stopped after 70 years since the clear cut had at this point been performed in all compartments and no further harvests could be carried out. Because the initial stand was of the age 40 years, the clear-cut forest area would require about 40 years to regrow to the same level and hence restore productivity at approximately year 110, while the selective thinning scheme would maintain a productive level throughout the entire simulation.

7.2 Simulation results

7.2.1 Stand size structure

After the initial stand has been generated and the trees' diameters transformed, the stand should exhibit a size structure (the composition of small and large trees) similar to the one shown in figure 7.2. As the simulations progress, however, the stand size structure is expected to diverge significantly between the low and the selective thinning simulations; before the clear cut, the low thinning simulation should experience an overall predominance of large trees, whereas the selective thinning simulation should experience a predominance of small and medium-sized trees.

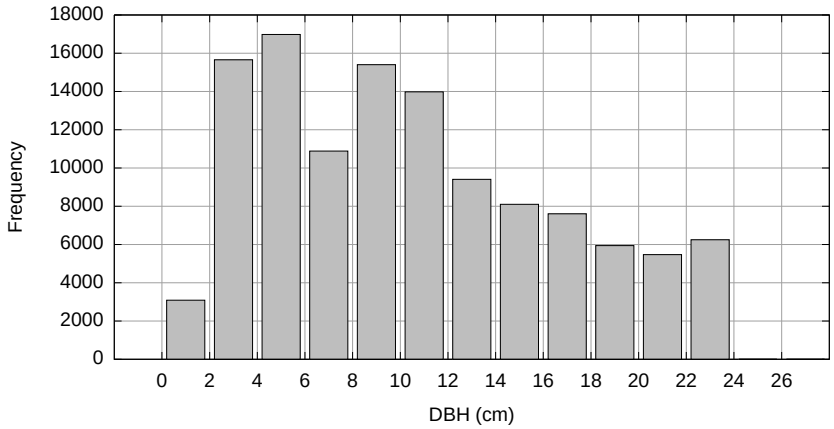


Figure 7.3: Resulting diameter distribution of the initial stand.

Figure 7.3 shows the stand structure of the initial stand after the transformation. The size structure of the trees in compartment 1 at specific time instances during the simulation can be seen in figure 7.4 for the low thinning scheme and figure 7.5 for the selective thinning scheme. The latter two figures present histograms over the diameter distribution at the end of two consecutive years, where a thinning is performed during the second year. Figure 7.4 shows a predominance of large trees with the exception being a large number of seedlings. The small trees have all been cut in the preceding low thinning and the seedlings have appeared during the subsequent years. Figure 7.5 on the other hand shows a wider range of tree sizes, spanning seedlings, saplings and more mature trees. Only the largest trees have been cut during the preceding selective thinning, thus allowing the smaller trees to progressively grow larger during the

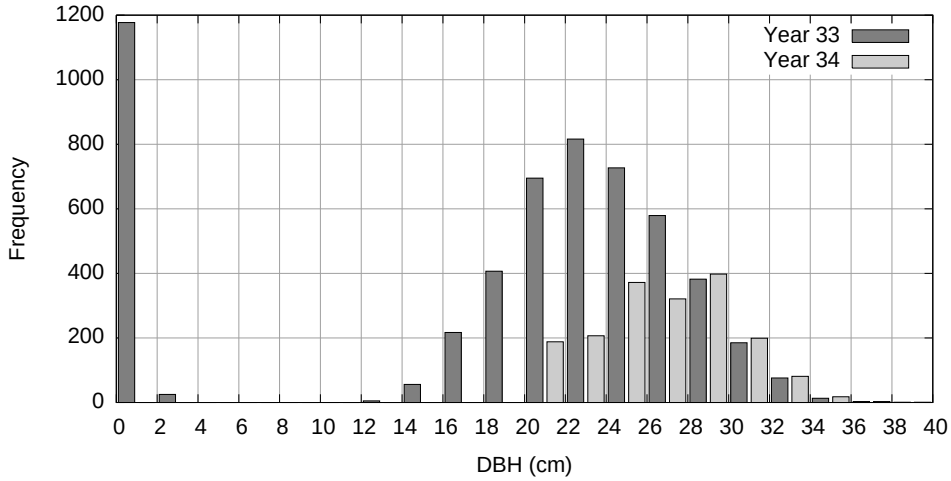


Figure 7.4: Diameter distribution at the end of year 33 and 34. A low thinning is performed during year 34.

subsequent years.

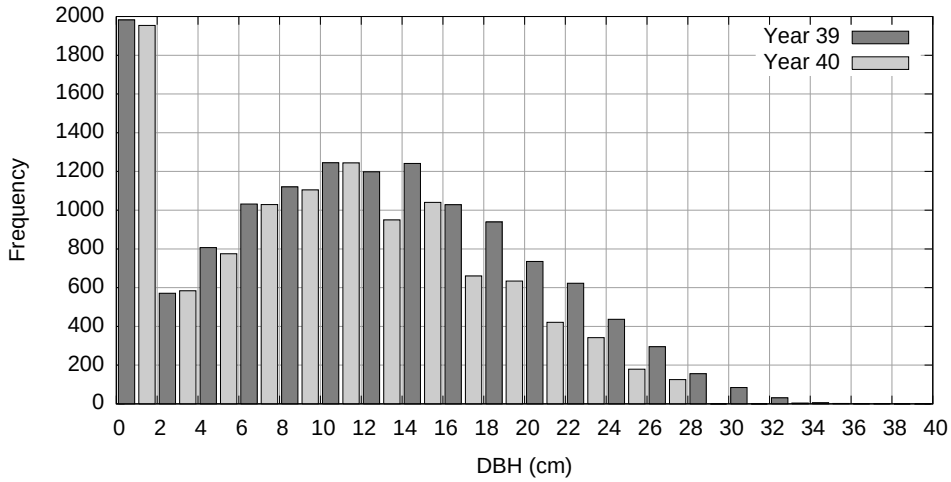


Figure 7.5: Diameter distribution at the end of year 39 and 40. A selective thinning is performed during year 40.

7.2.2 Growing stock

The growing stock of the simulated area should rise and fall repeatedly throughout the simulations as harvests are carried out. A drop in the growing stock volume is equal in magnitude to the volume of the trees removed in the corresponding harvest. The

growing stock development of the entire simulated area is shown in figure 7.6 both for the low and the selective thinning scheme. The jaggedness of the graph stems from the fact that different compartments did not reach the growing stock threshold of management methods simultaneously and hence performed harvests at different times.

The local effects of the thinning schemes on the growing stock can readily be visualised by plotting the growing stock development of a single compartment. The growing stock of compartment 1 is shown in figure 7.7. The small volume drops in the graph for the low thinning scheme correspond to the low thinnings, while the large drop, which reaches almost zero, corresponds to the final clear cut. The effects of the energy wood cutting, however, is not discernible. Similarly, the four volume drops for the selective thinning scheme corresponds to the four selective thinnings carried out.

7.2.3 Assortment volumes

The total assortment- and species-specific yield volumes are shown in table 7.2 and in figure 7.8. The low thinning scheme yielded in total 24,249 m^3, while the selective thinning scheme yielded 26,481 m^3. The selective thinning scheme produced slightly more sawlog and significantly more pulpwood, while the low thinning scheme produces slightly more energy wood. The species-composition was also different. The low thinning scheme produced more spruce wood, whereas the selective thinning scheme produced more birch wood.

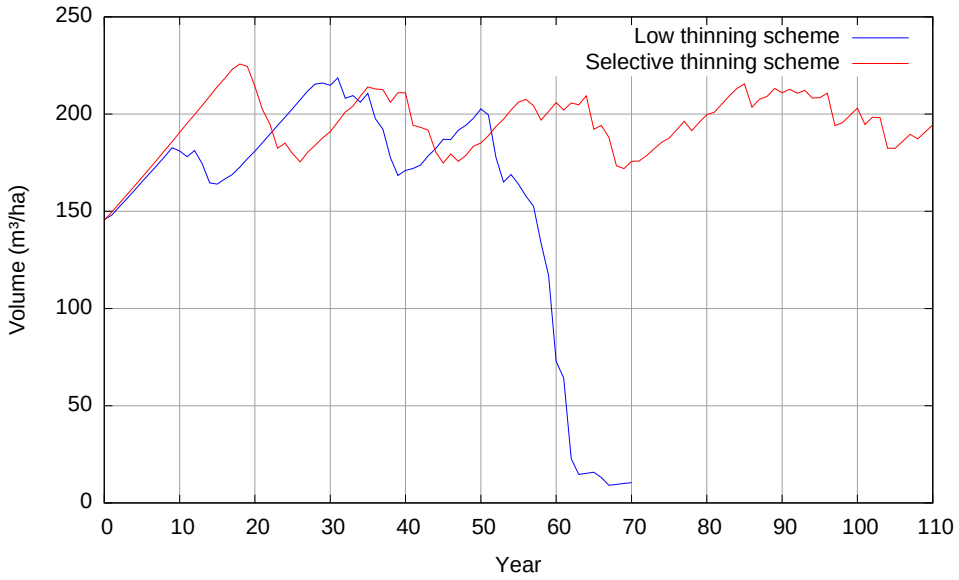


Figure 7.6: The growing stock of the entire simulated area throughout the simulation.

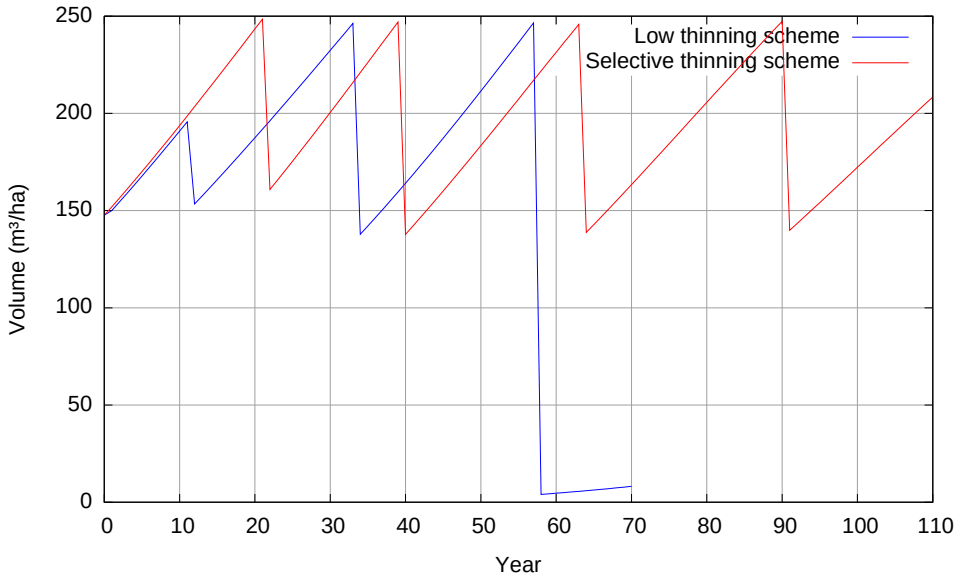
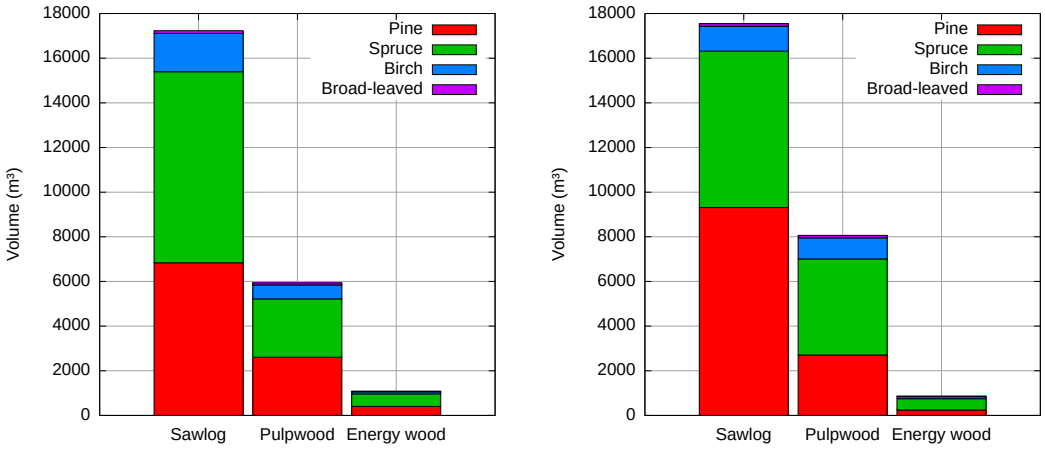


Figure 7.7: The growing stock of compartment 1 throughout the simulation.



(a) Low thinning scheme.

(b) Selective thinning scheme.

Figure 7.8: Total yield volume for different assortments and species.

Table 7.2: Total yield volume for different assortments and species

(m^3)	Pine	Spruce	Birch	Broad-leaved	Total
Low thinning scheme					
Sawlog	6,839	8,556	1,711	124	17,231
Pulpwood	2,609	2,608	628	84	5,929
Energy wood	406	556	94	32	1,089
Total	9,854	11,721	2,434	240	24,249
Selective thinning scheme					
Sawlog	9,310	7,006	1,116	116	17,548
Pulpwood	2,709	4,301	946	111	8,066
Energy wood	243	509	89	27	867
Total	12,261	11,815	2,151	254	26,481

8 CONCLUSIONS

Empirical research on forest growth and of the outcome and potential side effects of alternative silvicultural treatments and harvest methods may require several decades of studies. Utilising forest dynamics simulations in order to obtain these data within a feasible time frame can prove a viable alternative.

This thesis has described how computational statistical methods were implemented in SPATE-HPC. The theories behind the methods as well as the program implementations have been presented. The implementations were tested with a series of test cases, which showed that the implementations work well and produce the intended results.

A new, approximate algorithm based on the Cholesky decomposition for generating spatially correlated random numbers for environmental effects was implemented. The rationale for implementing the algorithm was to achieve scalability. The conducted performance and scalability tests showed that the algorithm indeed scales well and can be used to compute correlated random numbers for millions of trees, an achievement that cannot easily be accomplished with the traditional Cholesky decomposition. Furthermore, the error and correlation structure analyses indicated that the approximation error is negligible, especially when using an inclusion factor greater than 1.

A method for transforming tree characteristics of generated trees, formulae and algorithms for estimating tree trunk and wood assortment volumes, and low and selective thinning methods using Weibull distribution approximations were implemented. The individual functionalities of the methods as well as the synergy when combining the methods in a complete simulation were tested by conducting a case study. The study showed that the new functionalities were implemented properly and that they produce usable results.

On a general note, the study showed that it is possible with SPATE-HPC to simulate the development of a realistic forest stand and to obtain detailed statistics on forest growth and wood raw material volumes for different assortments and species. The study also demonstrated how one can use the simulator to compare the resulting yields

and possible side effects of different management schemes applied to the same forest area. One can therefore conclude that the implemented methods described in this thesis make SPATE-HPC a fully-functional forest simulator that can be used as a research platform for simulating real-world forest growth and harvest scenarios.

SPATE-HPC has been released as an open source product, and it, along with its documentation, is available for public download [2], thus making it possible for individuals, organisations and future research projects to utilise it freely. Although the simulator has been tested with Finnish tree species growing under Finnish climate conditions, no inherent design decision prevents one from implementing models for simulating completely different tree species growing in other parts of the world.

BIBLIOGRAPHY

[1] Krste Asanovic et al. A view of the parallel computing landscape. *Communications of the ACM*, 52(10):56–67, 2009.

[2] Suswood project web page. http://www.it.abo.fi/suswood/. Retrieved 2010-02-07.

[3] Chijien Lin. *Generating Forest Stands with Spatio-Temporal Dependencies*. PhD thesis, University of Joensuu, 2003.

[4] Magnus Södergård. *Development of an Object-Oriented Tree-Level Forest Simulator*. Master's thesis, Åbo Akademi University, 2008.

[5] Marc Snir et al. *MPI: The Complete Reference. Vol.1, The MPI Core*. The MIT Press, 2nd edition, 1998.

[6] Johan Schöring. *Design and Implementation of a Scalable Forest Simulator*. Master's thesis, Åbo Akademi University, 2008.

[7] Daryl J. Daily and David Vere-Jones. *An Introduction to the Theory of Point Processes: Elementary theory and methods*, volume 1. Springer, 2nd edition, 2003.

[8] Nicholas Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[9] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0. *W3C recommendation*, 6, 2000.

[10] Carlo Gaetan and Xavier Guyon. *Spatial Statistics and Modeling*. Springer, 2009.

[11] Richard L. Burden and J. Douglas Faires. *Numerical analysis*. Brooks/Cole, Pacific Grove (CA), 7th edition, 2001.

[12] Eunice E. Santos and Pei-Yue Chu. Efficient and optimal parallel algorithms for Cholesky decomposition. *Journal of Mathematical Modelling and Algorithms*, pages 217–234, 2004.

[13] Anshul Gupta and Vipin Kumar. A scalable parallel algorithm for sparse Cholesky factorization. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 793–802, New York, 1994. ACM.

[14] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. The MIT Press, 1999.

[15] Patrick P. A. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1):17–23, 1950.

[16] A. D. Cliff and J. K. Ord. Spatial and temporal analysis: autocorrelation in space and time. In Neil Wrigley and Robert John Bennett, editors, *Quantitative Geography: A British View*, pages 104–110. Routledge & Kegan Paul, London, 1981.

[17] CSC – IT Center for Science Ltd. Computing Servers. http://www.csc.fi/english/research/Computing_services/computing. Retrieved 2010-02-07.

[18] Douglas C. Montgomery and George C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Inc., 3rd edition, 2003.

[19] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, 1964.

[20] Waloddi Weibull. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 18(3):293–297, 1951.

[21] William H. Press et al. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[22] Touko Hyvämäki, editor. *Tapion Taskukirja*. Metsälehti Kustannus Oy, Helsinki, 2002.

[23] Jouko Laasasenaho. *Taper Curve and Volume Functions for Pine, Spruce and Birch*. Finnish Forest Research Institute, Helsinki, 1982.

[24] Hubert Hasenauer. *Sustainable forest management: growth models for Europe*. Springer, 2006.

SAMMANFATTNING

PROGRAMIMPLEMENTATION AV STATISTISKA METODER I EN SKALBAR TRÄDSIMULATOR

Introduktion

Skog utgör en viktig naturtillgång som kan användas för en rad olika ändamål. Alltefter-som marknadsefterfrågan förändras och ny miljölagstiftning träder i kraft kan skogs-ägare förväntas ta i bruk nya gallrings- och avverkningsmetoder. Eftersom träd växer långsamt är empiriska studier av skogstillväxt och effekten av alternativa avverknings-metoder en process som kan ta åtskilliga årtionden i anspråk. Datorbaserade simule-ringar kan således utgöra ett vettigt alternativ.

Denna avhandling beskriver arbetet kring trädsimulatorn SPATE-HPC. Simulatorn utnyttjar genomgående parallelliseringstekniker för att uppnå god skalbarhet och kun-na dra nytta av mångprocessorsystem. Syftet är således att kunna simulera stora skogs-arealer på tiotusentals hektar med miljontals träd. SPATE-HPC har utvecklats som en del av projektet SUSWOOD i samarbete mellan institutionen för informationstekno-logi vid Åbo Akademi och avdelningarna för geografi och forstvetenskap vid Joensuu Universitet, nuvarande Östra Finlands Universitet.

Syftet med denna avhandling är att beskriva programimplementationen av en upp-sättning statistiska metoder i SPATE-HPC som skrivits för att simulatorn ska uppfylla designkraven. Dessa metoder är: effektiv och skalbar generering av spatialt korrele-rade slumptal, fördelningstransformation av trädkarakteristika, stamvolymskattning, samt simulering av Weibullgallringsmetoder. Implementationerna testades också ge-nom en fallstudie som undersökte funktionaliteten hos de implementerade metoderna

samt deras samverkan när de tillämpades i ett realistiskt scenario.

Simulatorn

SPATE-HPC simulerar skogsutvecklingen som en iterativ process bestående av en serie tidssteg. Ett tidssteg, som kan motsvara ett eller flera år, utgörs av fyra oberoende faser: tillväxt, död, reproduktion och skötsel. I tillväxtfasen växer befintliga träd, i dödsfasen dör träd p.g.a. hög ålder, ofördelaktiga tillväxtförhållanden eller annan förstörelse, i reproduktionsfasen uppstår nya träd, och i skötselfasen utförs gallring och avverkning.

Varje träd simuleras individuellt i SPATE-HPC. Ett träd har en x-y–position, och dess storlek anges med kännetecknen diameter vid brösthöjd (1,3 meter ovanför markytan) samt höjd. Tillväxtmodellerna är utformade så att träd som står nära varandra påverkar varandra och tävlar sinsemellan om gemensamma resurser. Träden påverkas också lokalt av miljöeffekter, som kan innefatta torka eller insektsangrepp.

Då simuleringsprocessen startar kan det ursprungliga trädbeståndet laddas in från befintliga träddata, om sådana finns tillgängliga, eller så kan det genereras med en modell. Då simuleringen avslutats sammanställs statistik över beståndets utveckling och den utförda avverkningen.

Korrelerade slumptal

I tillväxtmodellerna i SPATE-HPC ingår miljöeffekter som uttrycks med spatialt korrelerade slumptal. Korrelationen är sådan att slumptal för träd som står närmare varandra korrelerar starkare än slumptal för träd som står längre ifrån varandra.

I tidigare versioner av simulatorn beräknades de korrelerade slumptalen genom att utföra en Cholesky-faktorisering av en kovariansmatris. Problemet med detta tillvägagångssätt är att man måste konstruera och upprätthålla en kvadratisk matris innehållande kovariansen mellan alla träd. Detta gör att programmets minnesanvändning växer kvadratiskt och dess körtid kubiskt i förhållande till skogens storlek – ett beteende som i praktiken förhindrar SPATE-HPC från att kunna simulera mer än några tiotusentals träd.

Istället utvecklades en approximativ metod som baserar sig på Cholesky-faktoriseringen. Denna metod, kallad den lokala Cholesky-faktoriseringen, utför istället många små Cholesky-faktoriseringar där enbart de närliggande träden beaktas. Me-

toden baserar sig således på antagandet att inverkan mellan träd som står långt ifrån varandra är liten och att felet som introduceras genom att ignorera dessa kan negligeras.

För att testa den numeriska noggrannheten samt metodens prestanda och skalbarhet utfördes en rad olika test. För små simuleringar kan den lokala Cholesky-faktoriseringen jämföras direkt med den traditionella algoritmen och ett exakt mått på approximationsfelet erhållas. För större simuleringar undersöktes korrelationsstrukturen genom att utföra ett antal simuleringar och sedan beräkna korrelationen från de simulerade datan. Prestandan och skalbarheten undersöktes också genom att progressivt förstora det simulerade skogsområdet samt öka antalet använda processorer och observera hur körtiden och minnesanvändningen påverkades.

Transformation av trädkarakteristika

Då nya träd genereras för det ursprungliga beståndet eller för reproduktionfasen så tilldelas träden kännetecknen diameter och höjd enligt en modell, varvid kännetecknen kommer att vara fördelade enligt en normalfördelning. För att man ska kunna generera träd med en helt godtycklig fördelning, vilket kan vara nödvändigt för att erhålla ett realistiskt trädbestånd, infördes en transformationsprocess för trädkarakteristika. Transformationsprocessen använder två kumulativa fördelningsfunktioner: den ena byggs upp från de genererade datan och den andra väljs av användaren. Genererade data översätts sedan från den ursprungliga till den önskade fördelningen så att den kumulativa sannolikheten för ursprungs- och resultatvärdena är densamma.

Fyra olika sätt att välja den önskade fördelningen implementerades. Användaren kan välja mellan en normalfördelning, en Weibullfördelning, en fördelning beräknad från empiriska datapunkter, samt en godtycklig empirisk fördelningsfunktion.

Volymskattning

En viktig fördel med att simulera skogstillväxt är att erhålla statistik om virkesvolymer. Både ett träds totala volym och volymen av vissa delsegment är användbara för att uppskatta totala virkestillgångar och skördesstorleken av olika typer av rundvirke.

Eftersom de enda attributen som beskriver ett träds storlek i SPATE-HPC är diametern vid brösthöjd och höjden, måste dess volym skattas med en matematisk modell.

En sådan modell som implementerats baserar sig på avsmalnande-kurvfunktioner, vilka ger en skattning av trädets diameter vid en godtycklig höjdposition. Volymen av godtyckliga delsegment kan således erhållas genom att numeriskt integrera diametern med avseende på höjdpositionen.

Förmågan att kunna beräkna volymen av delsegment användes sedan för att implementera volymskattning av olika typer av rundvirke som fås då fällda träd sågas till stockar. Exempel på sådant rundvirke är sågtimmer, massaved och energived. Olika rundvirkeskategorier har olika längd- och diameterkrav. SPATE-HPC beräknar virkesvolymerna genom att såga virtuella stockar från en trädstam, börjande från den grövsta änden av stocken med den första virkeskategorin. Programmet övergår sedan successivt till att såga stockar för den därpåföljande virkeskategorin vartefter stammen avsmalnar och inte längre kan satisfiera diameterkravet för den rådande kategorin.

Weibullgallring

I SPATE-HPC finns flera olika gallrings- och avverkningsmetoder implementerade. Två av dessa är låg- och urvalsgallring, vilka i SPATE-HPC kollektivt refereras till som Weibullgallring. Låg- och urvalsgallring är implementerade som stokastiska processer, d.v.s. träd fälls slumpmässigt enligt en sannolikhetsmodell. Låggallring fäller små träd, medan urvalsgallring fäller mestadels stora träd i täta omgivningar.

Weibullgallring baserar sig på att konstruera ett histogram enligt storleksklasser och att anpassa en Weibullfördelning på trädens storleksfördelning. Weibullfördelningens täthetskurva förskjuts sedan åt höger för låggallring och åt vänster för urvalsgallring. Weibullkurvan kommer således att beskära vissa histogramstaplar, och den del av staplarna som överstiger kurvan anger det antal träd som bör fällas i de motsvarande storleksklasserna.

Ett problem med detta tillvägagångssätt är att användaren inte vet hur mycket kurvan bör förskjutas, utan är i stället intresserad av att ange en avverkningsvolym som ska uppnås. Detta löstes genom att tillämpa en numerisk iterationsmetod som stegvis förskjuter Weibullkurvan tills en tillräckligt god resultatvolym erhållits. Som iterationsmetod valdes intervallhalveringsmetoden p.g.a. sin numeriska stabilitet och förmåga att alltid konvergera. Eftersom Weibullgallringen är en stokastisk process kan den därmed få andra oftast snabbare iterationsmetoder att oscillera våldsamt.

Fallstudie

För att testa implementationen av metoderna som diskuteras i denna avhandling samt deras samverkan utfördes en fallstudie. Studien bestod av två olika simuleringar av samma skogsområde. I den ena simuleringen tillämpades ett låggallringsschema och i den andra ett urvalsgallringsschema. Schemana var inrättade så att de skulle producera ungefär samma virkesvolym under motsvarande tidsperiod. Låggallringsschemat utförde först en energivedsgallring, sedan två låggallringar och till slut ett kalhygge. Urvalsgallringsschemat utförde i genomsnitt fyra urvalsgallringar med jämna intervall.

Simuleringarna visade att beståndsstrukturerna var fundamentalt olika för de båda schemana. I låggallringssimuleringen dominerade de stora träden medan i urvalsgallringssimuleringen förekom en bredare fördelning med både små och medelstora träd. Simuleringarna resulterade också i aningen olika virkesvolymer. Urvalsgallringsschemat producerade mer sågtimmer och massaved, medan låggallringsschemat producerade mer energived.

Resultat

Den approximativa algoritmen för att beräkna korrelerade slumptal testades på empirisk väg genom att utföra olika simuleringar som mätte både prestandan och den numeriska noggrannheten. Simuleringarna visade ett relativt litet approximationsfel som dessutom föll snabbt då antalet beaktade närliggande träd ökade. Simuleringarna visade också att den approximativa algoritmen, till skillnad från den traditionella, har en närapå linjär körtids- och minnesanvändningstillväxt, vilket gör den lämplig även för storskalig simulering.

Den separata fallstudien som undersökte implementationen av de statistiska metoderna visade att de är korrekt implementerade och producerar vettiga resultat. Studien demonstrerade också att man kan använda SPATE-HPC för att jämföra avkastningen och konsekvenserna av alternativa avverkningsscheman samt erhålla detaljerad statistik om skogstillväxt och virkesvolymer för olika virkestypskategorier.

SPATE-HPC har gjorts tillgänglig för allmänheten i form av öppen källkod. Privatpersoner och organisationer kan således använda och vidareutveckla programvaran och dra nytta av den i framtida forskningsprojekt.