

An Adaptive Computer Vision Technique for Estimating the Biomass and Density of Loblolly
Pine Plantations using Digital Orthophotography and LiDAR Imagery

by

Zachary J. Bortolot

Dissertation Submitted to the Virginia Polytechnic Institute and State University in Partial
Fulfillment of the Requirements of the Degree of

Doctor of Philosophy
in
Forestry

Dr. Randolph H. Wynne, Chairman

Dr. A. Lynn Abbott

Dr. James B. Campbell

Dr. Stephen P. Prisley

Dr. John R. Seiler

April 23, 2004
Blacksburg, Virginia

Keywords: computer vision, optimization, Nelder-Mead simplex, genetic algorithm, aerial
photograph, LiDAR, stand density, biomass, loblolly pine plantations

Copyright 2004, Zachary J. Bortolot

An Adaptive Computer Vision Technique for Estimating the Biomass of Loblolly Pine Plantations using Digital Orthophotography and LiDAR Imagery

Zachary J. Bortolot

Abstract

Forests have been proposed as a means of reducing atmospheric carbon dioxide levels due to their ability to store carbon as biomass. To quantify the amount of atmospheric carbon sequestered by forests, biomass and density estimates are often needed. This study develops, implements, and tests an individual tree-based algorithm for obtaining forest density and biomass using orthophotographs and small footprint LiDAR imagery. It was designed to work with a range of forests and image types without modification, which is accomplished by using generic properties of trees found in many types of images. Multiple parameters are employed to determine how these generic properties are used. To set these parameters, training data is used in conjunction with an optimization algorithm (a modified Nelder-Mead simplex algorithm or a genetic algorithm). The training data consist of small images in which density and biomass are known. A first test of this technique was performed using 25 circular plots (radius = 15 m) placed in young pine plantations in central Virginia, together with false color orthophotograph (spatial resolution = 0.5 m) or small footprint LiDAR (interpolated to 0.5 m) imagery. The highest density prediction accuracies (r^2 up to 0.88, RMSE as low as 83 trees / ha) were found for runs where photointerpreted densities were used for training and testing. For tests run using density measurements made on the ground, accuracies were consistently higher for orthophotograph-based results than for LiDAR-based results, and were higher for trees with DBH ≥ 10 cm than for trees with DBH ≥ 7 cm. Biomass estimates obtained by the algorithm using LiDAR imagery had a lower RMSE (as low as 15.6 t / ha) than most comparable studies. The correlations between the actual and predicted values (r^2 up to 0.64) were lower than comparable studies, but were generally highly significant ($p \leq 0.05$ or 0.01). In all runs there was no obvious relationship between accuracy and the amount of training data used, but the algorithm was

sensitive to which training and testing data were selected. Methods were evaluated for combining predictions made using different parameter sets obtained after training using identical data. It was found that averaging the predictions produced improved results. After training using density estimates from the human photointerpreter, 89% of the trees located by the algorithm corresponded to trees found by the human photointerpreter. A comparison of the two optimization techniques found them to be comparable in speed and effectiveness.

Acknowledgments

I am extremely grateful to many people and organizations for their assistance and support during my Ph. D. studies. First and foremost I would like to thank my fabulous advisor, Dr. Randolph Wynne. Randy was always patient and enthusiastic with my often cockamamie ideas, allowed me to chart my own path (while keeping me from wandering off cliffs!), and taught me a ton of stuff that went well beyond remote sensing.

I would like to thank my committee members for all of their insightful comments on the dissertation. In particular I would like to thank Dr. John Seiler for frequently reminding me that I am getting my degree in a forestry department, and therefore needed to cover my bases on forestry-related matters.

During my doctoral studies I received financial assistance from a number of sources, including the ASPRS Potomac Region, the International Society of Arboriculture, NASA, and the US National Guard. Without this support, getting my Ph. D. would have been much more difficult.

My studies were made much more enjoyable by the companionship of my fellow graduate students. In particular, I am very grateful to Christine Blinn, Beccy Musy, Sorin Popescu, and Jan van Aardt for forcing me to do things other than work.

Jan van Aardt, Gleb Tcheslavski, Jeff Bardwell, and Troy Wasky were huge helps while I was collecting field data. If they had not braved the green briar and scorching temperatures it would have taken twice as long to get half as much data.

While getting my Ph. D., my parents, wife, and brother all gave me huge amounts of moral support and put up with all the negative effects getting a Ph. D. has on family affairs. In particular I would like to thank my parents for their frequent e-mails and my wife for her frequent telephone calls.

Finally, I would like to thank my optimization algorithms. Optimization is really everywhere in life, and spending hours watching optimization take place really gave me insight into how the world works.

Table of contents

Chapter 1: Introduction	1
1.1 Carbon dioxide and forests	1
1.2 Quantifying the ability of forests to remove carbon from the atmosphere	2
1.3 Estimating forest carbon sequestration using remotely sensed imagery	3
1.3.1 Radar estimation of biomass	6
1.3.2 Estimation of biomass using statistically-processed passive optical remote sensing data	14
1.3.3 Estimation of biomass from high resolution passive optical imagery using object-oriented approaches	14
1.3.4 Light detection and ranging (LiDAR) estimating of biomass	16
1.3.5 Summary of techniques used for remotely estimating stand or tree biomass	18
1.3.6 Justification for using aerial photography and small footprint LiDAR data and an object-oriented approach	19
1.3.6.1 Advantages and disadvantages of high spatial resolution passive optical and small footprint LiDAR imagery	20
1.3.6.2 Comparison between object-oriented and non-object-oriented approaches	24
1.3.6.3 Weighing the advantages and disadvantages	26
1.4 Summary and the objectives of the dissertation	27
Chapter 2: Literature review	29
2.1 Introduction	29
2.2 Past methods for using object-oriented computer vision techniques for locating trees in high resolution imagery and delineating their crowns	30
2.3 Results of past studies	33
2.3.1 The valley following approach	33
2.3.2 Crown identification using curved edge segments	34
2.3.3 Template matching	34
2.3.4 Peak finding methods	35
2.3.5 Segmentation method	38
2.3.6 Comparison and criticism of these results	38
Chapter 3: Methods	39
3.1 Overview	39
3.2 Image processing algorithm	39
3.2.1 Generic attributes	39
3.2.2 An algorithm that uses these generic attributes to locate trees and find their boundaries	41
3.2.3 Parameter descriptions and initial values	63

3.2.4 Training	67
3.2.4.1 Optimization techniques	72
3.2.4.1.1 The Nelder-Mead simplex algorithm	72
3.2.4.1.2 A genetic algorithm	79
3.3 Study site, available data, and data preprocessing	83
3.4 Ground data collection	85
3.5 Biomass estimation	89
3.6 Evaluating the algorithm	90
 Chapter 4: Results	 98
4.1 Overview	98
4.2 Derivation of equation forms	98
4.3 Analysis of the computer vision algorithm after training using all 25 datasets	99
4.3.1 Residual analysis	99
4.3.2 Intercomparison of results obtained using all the plots as training and testing data	106
4.3.2.1 Intercomparison of tree count results	106
4.3.2.2 Intercomparison of biomass results	109
4.3.3 Comparison to human photointerpreter	109
4.3.3.1 Determining whether the same trees were found	109
4.3.3.2 Comparison of ground count predictions	116
4.4 Analysis of the computer vision algorithm after training and testing using subsets of the data	117
4.4.1 Effects of training sample size on accuracy	117
4.4.1.1 Effect on tree count accuracies	117
4.4.1.2 Effect on biomass prediction accuracies	117
4.4.2 Comparison of the two training sets	125
4.4.2.1 Comparison of tree count results of the two datasets	125
4.4.2.2 Comparison of biomass results for the two datasets	129
4.4.3 Evaluation of methods of postprocessing the results	129
4.5 Comparison with results from other studies	135
4.5.1 Comparison of tree location and density results	137
4.5.2 Comparison of biomass results	135
4.6 Comparison of the Nelder-Mead simplex and genetic algorithm techniques	138
 Chapter 5: Conclusions	 145
5.1 Summary and discussion on results	145
5.2 Caveats	147
5.3 Future research	147
 References	 149

Appendix A. The Nelder-Mead simplex based version of the program for setting paramaters for the program that located and measures trees in orthophotographs	164
Appendix B. The genetic algorithm based version of the program for setting paramaters for the program that located and measures trees in orthophotographs	191
Appendix C. A program for using the parameters found by the program in Appendix A or B to determine the number of trees in test images.	220
Appendix D. The program for setting paramaters for the program that locates trees in LiDAR images and simultaneously determines biomass.	234
D.1 Main program	234
D.2 Module nelder_biomass_score.py	261
Appendix E. The program for setting paramaters for the program that locates trees in LiDAR images and does not simultaneously determines biomass.	270
Appendix F. A program for using the parameters found by the program in Appendix D or E to determine the number of trees and biomass in test images.	297
Appendix G. Measurements for the trees in each plot whose DBH and dominance status were measured.	314
Curriculum vitae	339

List of figures

Figure 1. Three dimensional diagrams constructed from corresponding images of a pine plantation	32
Figure 2. Orthophotograph and LiDAR images of a pine plantation and corresponding ground photographs	40
Figure 3. A flow chart showing the basic steps used by the computer vision algorithm described in Section 3.2.2 for processing a digital aerial photograph or other high resolution passive optical image	42
Figure 4. A flow chart showing the basic steps used by the computer vision algorithm described in Section 3.2.2 for processing a canopy height model (CHM) derived from small footprint LiDAR imagery	43
Figure 5. Steps of the tree finding algorithm (aerial photograph)	44
Figure 6. Steps of the tree finding algorithm (LiDAR CHM)	45
Figure 7. The Laplacian of the Gaussian used for finding local minima	50
Figure 8. The processing flow used for processing orthophotographs or other high spatial resolution passive optical data, using training data to set the parameters for the computer vision algorithm	68
Figure 9. The processing flow used for processing LiDAR CHMs, using training data to set the parameters for the computer vision algorithm. In this case the biomass of the stands are determined using a regression equation that is derived after training is complete (the first biomass determination procedure described in Section 3.2.2)	69
Figure 10. The processing flow used for processing LiDAR CHMs, using training data to set the parameters for the computer vision algorithm. In this case the biomass of individual trees is determined and summed (the second biomass determination procedure described in Section 3.2.2)	70
Figure 11. The difference between local and global optimization	73
Figure 12. A representation of the simplex used for simultaneously optimizing two parameters	74
Figure 13. A flow chart showing the Nelder-Mead simplex algorithm	76

Figure 14. A visual depiction of a well fit model and an overfit model	79
Figure 15. A map showing the location of the Appomattox-Buckingham State Forest	83
Figure 16. Map showing the locations of the study plots overlain on a stand map of the Appomattox-Buckingham State Forest	88
Figure 17. The relationship between the number of trees in plots 1 - 11 for which DBH and dominance status were measured, and the uncertainty in the biomass estimate for trees with DBH ≥ 7 cm	91
Figure 18a. Residuals when photointerpreted tree counts in all plots were used for training and testing	101
Figure 18b. Residuals when the number of trees with DBH ≥ 7 cm in all plots were used for training and testing together with the orthophotograph	101
Figure 18c. Residuals when the number of trees with DBH ≥ 10 cm in all plots were used for training and testing together with the orthophotograph	102
Figure 18d. Residuals when the number of trees with DBH ≥ 7 cm in all plots were used for training and testing together with the LiDAR CHM. Biomass was not simultaneously optimized	102
Figure 18e. Residuals when number of trees with DBH ≥ 10 cm in all plots were used for training and testing together with the LiDAR CHM	103
Figure 18f. Residuals when biomass of trees with DBH ≥ 10 cm in all plots were used for training and testing together with the LiDAR CHM	103
Figure 19. LiDAR images produced after training using all 25 training plots and no biomass data	104
Figure 20. A plot showing the actual versus the predicted number of trees for the most accurate solution set obtained when training was conducted using ground based tree counts of all 25 plots (DBH ≥ 7 cm) and the corresponding air photo imagery . . .	108
Figure 21. A plot showing the actual versus the predicted biomass trees for the most accurate solution set obtained when training was conducted using ground biomass estimates of all 25 plots (DBH ≥ 10 cm) and the corresponding LiDAR imagery	110
Figure 22a - e. Aerial photograph segments corresponding to plots 1 (upper left), 2 (upper center), 3 (upper right), 4 (middle left), and 5 (lower left)	111

Figure 22f - j. Aerial photograph segments corresponding to plots 6 (upper left), 7 (upper center), 8 (upper right), 9 (middle left), and 10 (lower left)	112
Figure 22k - o. Aerial photograph segments corresponding to plots 11 (upper left), 12 (upper center), 13 (upper right), 14 (middle left), and 15 (lower left)	113
Figure 22p - t. Aerial photograph segments corresponding to plots 16 (upper left), 17 (upper center), 18 (upper right), 19 (middle left), and 20 (lower left)	114
Figure 22u - y. Aerial photograph segments corresponding to plots 21 (upper left), 22 (upper center), 23 (upper right), 24 (middle left), and 25 (lower left)	115
Figure 23. Plots constructed using the biomass of trees with $DBH \geq 10$ cm, and predictions of biomass made when the algorithm was optimized using all 25 plots and the biomass equation was derived after optimization	124
Figure 24. Plots constructed using the photointerpreted tree counts, and predictions of tree counts made when the algorithm was optimized using all 25 plots	126
Figure 25. Plots constructed using the ground tree counts of trees with $DBH \geq 7$ cm, and predictions of tree counts made when the algorithm was optimized using all 25 plots and the orthophotograph	127
Figure 26. Plots constructed using the ground tree counts of trees with $DBH \geq 7$ cm, and predictions of tree counts made when the algorithm was optimized using all 25 plots and the LiDAR CHM	128
Figure 27. Comparison of the Nelder-Mead simplex and genetic algorithm optimization techniques. Optimizations were run for 5000 evaluations using photographs and ground count data ($DBH \geq 7$ cm) from three plots	141
Figure 28. Comparison of the Nelder-Mead simplex and genetic algorithm optimization techniques. Optimizations were run for 5000 evaluations using photographs and ground count data ($DBH \geq 7$ cm) from twelve plots	142
Figure 29. Comparison of the Nelder-Mead simplex and genetic algorithm optimization techniques. Optimizations were run for 5000 evaluations using photographs and ground count data ($DBH \geq 7$ cm) from 25 plots	143

List of tables

Table 1. Price ranges for data collected by selected sensors	6
Table 2. Summary of studies that have used remotely sensed data to estimate forest biomass or volume	7
Table 3. Results obtained by Wulder <i>et al.</i> (2000) using panchromatic data and either fixed window sizes or variable window sizes calculated using the semivariance (SVR) or slope break (SB) technique	36
Table 4. Independent variables considered in the stepwise multiple linear regression used to create equations for the first biomass prediction technique described in Section 2.3.3.2	61
Table 5. System for determining ξ_j , which is used for determining the starting vertices for the simplex.	74
Table 6. Attributes of the plots used for the study	86
Table 7. Coordinates of plots used for the study	87
Table 8. Survey of equations developed for predicting the biomass of aboveground loblolly pine components	92
Table 9. The plots that were selected as testing data (T), training data used in the optimization (Y), and training data not used in optimization (E)	94
Table 10. Tests performed for evaluating the algorithm	95
Table 11. Postprocessing techniques that were evaluated	96
Table 12. Tree count results obtained when all 25 plots were used for testing and training . .	100
Table 13. Biomass results obtained when all 25 plots were used for testing and training	100
Table 14. Durbin-Watson statistics for the runs that were trained and tested using all plots . .	105
Table 15. Accuracies of tree counts obtained by a human interpreter, based on ground counts from all 25 plots	116

Table 16. Tree count results obtained using subsets of the data	118
Table 17. Individual plot density estimates obtained using six training images (training set 1)	120
Table 18. Biomass results obtained from the LiDAR imagery using subsets of the data for training and testing	123
Table 19. Correlation coefficients obtained for tree count predictions using the postprocessing techniques described in Table 11	130
Table 20. Correlation coefficients obtained for LiDAR derived biomass predictions using the postprocessing techniques described in Table 11	132
Table 21. RMSE values (in trees / ha) obtained for tree count predictions using the postprocessing techniques described in Table 11	133
Table 22. RMSE values (in t / ha) obtained for LiDAR-based biomass predictions using the postprocessing techniques described in Table 11	134
Table 23. Overall effectiveness of the postprocessing methods in terms of correlation coefficient (Δr) and RMSE ($\Delta RMSE$)	134
Table 24. Studies that used remote sensing techniques to determine the aboveground biomass of forest stands	139

List of procedures

Procedure 1. Pseudocode for the vegetation index creation procedure	46
Procedure 2. Pseudocode for a procedure that performs a 3x3 low pass filter on a single band image, <i>image</i> , to produce a filtered image, <i>lp_image</i>	47
Procedure 3. Pseudocode for using LoG filters to obtain local minimum (<i>lmin_image</i>) and local maximum (<i>lmax_image</i>) images based on a single band input image, <i>image</i>	48
Procedure 4. Pseudocode for the orthophotograph-based form of a function that locates possible trees in an image and attempts to find their diameters	53
Procedure 5. Pseudocode for the LiDAR-based form of a function that locates possible trees in an image and attempts to find their diameters	55
Procedure 6. Pseudocode for the orthophoto-based form of the procedure of eliminating trees found by Procedure 4 that are unlikely to correspond to actual trees	57
Procedure 7. Pseudocode for the LiDAR-based form of the procedure of eliminating trees found by Procedure 5 that are unlikely to correspond to actual trees	59

Chapter 1

Introduction

1.1 Carbon dioxide and forests

Since 1850, humans have released approximately 480 Gt of CO₂ into the atmosphere through fossil fuel burning and land use changes. This has led to atmospheric CO₂ concentrations that are higher than at any other point during the past twenty million years (Malhi *et al.*, 2002). The United States has contributed a significant portion of this CO₂, and in 2001 the US emitted 6937.7 Tg of CO₂, an increase of 13.0% over the 1990 level (EPA, 2003, p. ES2 - ES3). The high levels of atmospheric CO₂ are very important to our society since the increased atmospheric CO₂ concentration may dramatically affect many aspects of the earth. Climatologically, increased CO₂ levels are a danger because CO₂ acts as a greenhouse gas. Although the warming potential of CO₂ is small compared to that of other gases such as water vapor, nitrous oxide, and methane, the large quantities of CO₂ emitted by fossil fuel consumption contribute 60% of the humanly induced warming potential (Gates, 1993, p. 8-10). Global warming caused by the increase in greenhouse gases may lead to ecosystem changes, precipitation changes, sea level rises and a loss of biodiversity (Nilsson, 1992, p. 83-84, 98-99, 162-165). In addition to climatic effects, the increased CO₂ levels have been demonstrated to directly impact the environment by decreasing the advantage C4 plants have over C3 plants (Johnson *et al.*, 1993) and by increasing plant growth rates (Nilsson, 1992, p. 111-113).

Forests play an important role in the global carbon cycle, since each year forests absorb approximately one twelfth of the earth's atmospheric CO₂ stock (Malhi *et al.*, 2002), and much of this carbon is stored as woody biomass or cycled into the soil. Overall, forested ecosystems account for approximately 72% of the earth's terrestrial carbon storage (Malhi *et al.*, 2002). In the US in 2001, non-urban forests covered 33% of the area of the US and sequestered 759.0 Tg of carbon in 2001. Urban forests covered 3.5% of the area of the US sequestered 58.7 Tg of

carbon. These respectively represent 10.9% and 0.8% of the total US emissions (EPA, 2003, p. 149-158).

Because forests play such a large role in the carbon cycle, two techniques have been proposed to manage forests in ways that will lead to a reduction in atmospheric CO₂. The first approach is to reforest deforested or marginal land, since the new forests will absorb CO₂ through the process of photosynthesis and store it as biomass (Breuer, 1979; Dyson and Marland, 1979; Houghton and Woodwell, 1989; Houghton, 2002). Houghton and Woodwell (1989) estimate that five million acres of deforested land exist in the tropics, and that this number could be increased by a further 3.5 million square kilometers if farmers in this area were to switch from shifting to fixed agriculture. If the 8.5 million square kilometers were planted with trees, the trees would be able to offset one billion of the five billion tonnes of carbon humans emit yearly. During the 1990s, it is estimated that 490 Tg / yr of carbon were removed from the atmosphere from afforestation and reforestation (Houghton, 2002).

The second, and more easily implemented, means of removing CO₂ is by maintaining existing forests, but storing the wood from these forests in an environment with a slow decomposition rate (Dyson, 1977; Dyson and Marland, 1979; Cooper, 1983). Although these authors have suggested abandoned mine shafts and bogs as possible sites for storing the wood, the same effect can be achieved in a more economically feasible manner by turning the wood into products which will undergo slow decomposition rates. It is estimated that 2.8 Pg of carbon is currently stored in wood products in the US, and at least 4 Pg of carbon is stored in the northern hemisphere exclusive of China (Goodale *et al.*, 2002).

1.2 Quantifying the ability of forests to remove carbon from the atmosphere

In order to quantify the ability of a forest to remove CO₂ from the atmosphere, a parameter known as the net primary productivity (NPP) is usually calculated. Although NPP can be defined in multiple ways, in relation to the carbon cycle it is defined as the net amount of carbon accumulated by one or more plants in a fixed period of time (Liu *et al.*, 1999). There are two common means of calculating NPP: Using the difference in carbon storage at two periods of time and using forest process models. Knowing the forest biomass is generally a necessary

component of both. The difference method involves the following steps: 1) Determining the amount of biomass at two or more points in time, 2) converting from biomass to carbon (generally by multiplying by a fraction such as 0.5) and 3) using subtraction and division operations to determine the change in carbon on a yearly basis. This technique has been successfully used by several studies. Recent studies using this approach include the work by Lodhiyal and Lodhiyal (1997) to determine the NPP of trees, herbs and shrubs in a poplar (*Populus deltoides*) plantation in India and Riley and Phillips' (1997) attempt to determine the landscape-level NPP of the Los Tuxtlas region in southeastern Mexico. In an urban environment, this approach has been used by Nowak (1994) to determine the NPP for urban forests in Chicago, by McPherson (1998) for determining the NPP of Sacramento's urban forests, and for ten cities by Nowak and Crane (2002). Jo and McPherson (1995) used this approach to look at the system-wide NPP for a two block section of Chicago.

Process models that attempt to simulate many tree, stand and / or landscape factors affecting NPP have become increasingly popular tools for quantifying NPP and making predictions of future changes in NPP. Examples of these models include StandCarb (Cohen *et al.*, 1996; Harmon and Domingo, 2001), PnET (Aber and Federer, 1992), Biome-BGC (Running and Coughlin, 1988; Running and Hunt, 1993), BIOMASS (McMurtrie *et al.*, 1992) and 3-PG (Landsberg and Waring, 1997). Several of these models (StandCarb, Biome-BGC, and 3-PG) either require or can incorporate biomass information, and biomass values can be used to verify the output from these models.

1.3 Estimating forest carbon sequestration using remotely sensed imagery

In general, both national and regional / local scale forest carbon sequestration estimates are obtained using ground based techniques. Although ground based techniques have enabled estimates to be obtained in many areas, these approaches do have limitations. The first is that they are often imprecise due to the small portion of the trees that are sampled. Phillips *et al.* (2000) analyzed the errors associated with carbon sequestration estimates based on Forest Inventory and Analysis (FIA) data and found that errors in volume change between inventories (which are likely to be very similar to the errors in carbon storage change) were approximately

20% for regional estimates, and varied from 12% to 139% for state-level estimates. Ninety to 99% of these errors resulted from sampling errors due to measuring only a subset of the area of interest.

The second limitation is that timely data are difficult to collect due to the cost and data collection times associated with ground inventories. For example, until the switch to an annual inventory in 1998, FIA data were collected for the same area only once every five to ten years (USFS, 2001), yet the amount of carbon sequestered by non-urban forests in the U. S. changed by approximately 22.8% between 1990 and 2001 (EPA, 2003, p. 149). These dramatic changes, combined with the difficulty in modeling changes in forest carbon storage, mean that it is difficult to obtain accurate, timely estimates of carbon storage.

A third limitation of ground based studies is cost. In order to achieve accurate measurements of carbon sequestration rates, many plots must be established. This is especially true in areas with highly heterogeneous forests, such as those found in the tropics (Boscolo *et al.*, 2000). By having such high accounting costs, it is possible that the cost of inventorying the data will make forests a less attractive atmospheric CO₂ mitigation strategy due to economic reasons (Boscolo *et al.*, 2000).

A final limitation is that private landowners may restrict access to their lands, preventing ground-based inventories from being obtained. Although this is not a serious issue for FIA data collection, it may play a large role in other data collection efforts.

The limitations of sample size, timeliness, expense, and access can potentially be addressed at a range of scales by using remote sensing techniques to estimate biomass. Several remote sensing techniques have been used to successfully estimate biomass (see Sections 1.3.1 to 1.3.5). These include high spatial resolution passive optical (spatial resolution ≤ 10 m), medium resolution passive optical ($10\text{m} \leq \text{spatial resolution} \leq 100$ m), radar, and small and large footprint light distance and ranging (LiDAR) data, which are a type of active optical data. Most remotely sensed data are sampled continuously or with a very tight grid spacing, which means that it is possible to obtain data for every point in an area of interest. This directly addresses the issue of uncertainty in forest biomass estimates due to sampling error, because it is no longer necessary to use statistical extrapolation to determine the biomass of areas that were not sampled.

Timeliness can also be addressed by using remote sensing. It takes much less time to process an image than it does to manually measure (*in situ*) the area covered by the image. As an example, physically counting the number of trees in a circular plot (diameter = 15m) in an extensively managed loblolly pine plantation takes approximately 30 to 90 minutes, whereas after training, the algorithm described in Chapter 3 can process an orthophotograph of the same area in under five seconds on a computer using a 2.4 GHz Intel Xeon processor. Additionally, it is possible to obtain imagery for a very large area at a single point in time, reducing the temporal discontinuities among data from different portions of the study area.

Many types of remotely sensed data are quite inexpensive, as seen in Table 1, which lists the price per square kilometer for several types of imagery. Performing a ground inventory for carbon can cost considerably more. Boscolo *et al.* (2000) reported that it cost approximately \$55 / km² to conduct a ground based carbon inventory in Bolivia, and Brown *et al.* (2000) reported costs of \$100 and \$500 / km² in Costa Rica and India respectively. Similar inventories in the U.S. would probably cost much more due to higher labor costs. These rates are higher than the costs associated with satellite-borne remote sensing data types that have been successfully used for biomass determination. In the developed world, aircraft-borne remote sensing data types may also be less expensive than ground studies.

Several court cases have examined the legality of obtaining imagery of private property. The rulings state that it is legal to acquire and use the imagery, so long as it does not reveal subsurface properties of the object being examined (e.g., *Dow Chemical Co. v. United States*, *supra*, 476 U.S. 227, 106 S.Ct. 1819, 90 L.Ed.2d 226 (1986) and *U.S. v. Field*, 855 F. Supp. 1518 (W.D. Wis. 1994)). This clearly indicates that aerial photographs and optical satellite data can be acquired and used over all private property. However, the legality of radar and LiDAR data may be open to challenge because they are able to return information about the subsurface.

Table 1. Price ranges for data collected by selected sensors.

Sensor name	Platform	Sensor type	Cost (/ km ²)	Source
Aerial photographs	Aircraft	High resolution color	\$193 (approximate)	Czaplewski (1998)
LiDAR	Aircraft	Small footprint LiDAR	\$700 (approximate)	I. Wosiski, Airborne 1 Corp., Pers. Comm., 4/10/2003
QuickBird	Satellite	High resolution panchromatic and multispectral	\$22.50 (panchromatic or multispectral)	DigitalGlobe (2003)
Ikonos	Satellite	High resolution panchromatic and multispectral	\$7 to \$29	Space Imaging (2001)
Landsat ETM+	Satellite	Medium resolution multispectral	\$0.15 to \$0.19	USGS (2002)
SPOT	Satellite	Medium resolution multispectral, high resolution panchromatic	\$0.38 to \$19.53	Spot Image (2002) based on currency conversion rate on January 31, 2004
ERS-1 and 2	Satellite	Radar	\$0.09 to \$0.16	Eurimage (2003)
Radarsat	Satellite	Radar	\$0.12 to \$1.50	Radarsat International (2002)

1.3.1 Radar estimation of biomass

Over the past decade, many studies have successfully used radar data to determine forest biomass. The traditional approach for processing the radar data has been to develop models in which the dependent variable is the forest biomass quantity of interest, and the independent variables are the average radar backscatters for the area of interest, often measured at different polarizations and frequencies (see Table 2). This method has been shown to be very effective for pine (Wu, 1987; Hussin *et al.*, 1991; Le Toan *et al.*, 1992; Dobson *et al.*, 1992) and hardwood (Isrealsson *et al.*, 1995) plantations. Pulliainen *et al.* (1994) reported moderate success using a scatterometer to determine tree volume of naturally growing trees in a boreal forest. Dobson *et*

Table 2. Summary of studies that have used remotely sensed data to estimate forest biomass or volume. In studies in which both volume and biomass were estimated, only the biomass results are reported. The results reflect the best models developed by the authors. Studies that predicted biomass and volume have been colored yellow and blue respectively.

Study	Sensor type	Sensor name	Dependent variable	Independent variables	Forest type	r ²	RMSE	RMSE (% of average)
Wu (1987)	Radar	SIR-A	Aboveground biomass	Backscatter	Pine plantation (species not reported)	0.83	31 t / ha	30
Hussin <i>et al.</i> (1991)	Radar	NASA / Ames CV-990 Airborne Laboratory	Aboveground biomass	Backscatter	Slash pine plantation	0.98	1 t / ha	0
Le Toan <i>et al.</i> (1992)	Radar	AirSAR	Stem biomass	Backscatter	Maritime pine plantation	0.95	Not reported	Not reported
Isrealsson <i>et al.</i> (1995)	Radar	MAESTRO-1	Stem volume	Backscatter	Poplar (various varieties) and European ash plantation	0.78 (poplar), 0.95 (ash)	Not reported	Not reported
Pulliainen <i>et al.</i> (1994)	Radar (scatterometer)	HUTSCAT	Stem volume	Backscatter	Boreal	0.56	Not reported	Not reported
Hyppa <i>et al.</i> (2000)	Radar	ERS-1/2	Stem volume	Coherence	Boreal	0.24	91 m ³ / ha	58
Hyppa <i>et al.</i> (2000)	Radar	ERS-1/2	Stem volume	Backscatter, filtered backscatter, mean and standard deviations over multiple dates, differences between multiple dates	Boreal	0.06	102 m ³ / ha	65
Hyppä <i>et al.</i> (2000)	Radar	JERS-1	Stem volume	Backscatter, first three principal components of three images, differences between multiple dates, image ratios	Boreal	0.13	98 m ³ / ha	63
Hyppä <i>et al.</i> (2000)	Radar	HUTSCAT	Stem volume	Mean and weighted mean profile heights, ground and crown backscatters (HH, VV, VH)	Boreal	0.68	55 m ³ / ha	34

Study	Sensor type	Sensor name	Dependent variable	Independent variables	Forest type	r ²	RMSE	RMSE (% of average)
Fransson <i>et al.</i> (2001)	Radar	ERS-1/2	Stem volume	Coherence	Boreal	0.86	26 m ³ /ha	20
Pulianent <i>et al.</i> (2003)	Radar	ERS-1/2	Stem volume	Coherence (multitemporal)	Boreal	0.81	Not reported	48
Lefsky <i>et al.</i> (1999a)	Large footprint LiDAR	SLICER	Aboveground biomass	Canopy surface height and canopy volume method indices	Closed canopy Douglas-fir / western hemlock	0.91 (adjusted)	Not reported	Not reported
Lefsky <i>et al.</i> (1999b)	Large footprint LiDAR	SLICER	Aboveground biomass	Canopy height indices	Deciduous forest in Maryland	0.80	Not reported	Not reported
Lefsky <i>et al.</i> (2001)	Large footprint LiDAR	SLICER	Aboveground biomass	Max / min canopy height, canopy cover, variability in the upper canopy surface, total volumes of foliage and empty space in canopy	Natural Douglas-fir / western hemlock	0.86 (adjusted)	164 t / ha	24
Lefsky <i>et al.</i> (2002)	Large footprint LiDAR	SLICER	Aboveground biomass	Canopy height indices	Temperate deciduous, temperate coniferous, boreal	0.56 (temp. decid.), 0.65 (temp. conif.), 0.87 (boreal), 0.84 (all)	3 t / ha (temp. decid.), 8 t / ha (temp. conif.), 29 t / ha (boreal), 7.5 t / ha (all)	13 (temp. decid.), 3 (temp. conif.), 5 (boreal), 2 (all)
Drake <i>et al.</i> (2002)	Large footprint LiDAR	LVIS	Aboveground biomass	LiDAR canopy height, height of median energy, height / median ratio, ground return ratio	Tropical wet forest	0.73	60 t / ha	49

Study	Sensor type	Sensor name	Dependent variable	Independent variables	Forest type	r ²	RMSE	RMSE (% of average)
Means <i>et al.</i> (1999)	Large footprint LiDAR	SLICER	Aboveground biomass	LiDAR canopy height, quadratic mean canopy height, canopy reflectance sum	Douglas-fir / western hemlock	0.96	88 t / ha	18
Nelson <i>et al.</i> (1988)	Small footprint LiDAR	Custom built laser profiler	Total tree biomass	Average of the three greatest laser heights, mean plot height (all pulses and canopy pulses), distance between the top of canopy and a point 2, 5 or 10 m above ground	Pines (including plantations) and hardwoods	0.55	68 t / ha	Not reported
Naesset (1997)	Small footprint LiDAR	Optech ALTM 1020	Stem volume	Mean stand height, canopy cover density	Norway spruce and Scots pine	0.89	26 m ³ /ha	18
Hyypä and Inkinen (2002)	Small footprint LiDAR	TopoSys-1	Stem volume	Individual tree height	Norway spruce and Scots pine	0.98	17 m ³ /ha	10
Popescu <i>et al.</i> (2003), Popescu and Wynne (in press)	Small footprint LiDAR	Aeroscan	Aboveground biomass	Number of trees, average / maximum crown width, average / maximum height	Pines and hardwoods	0.82 (pines) / 0.33 (hardwoods)	29 t / ha (pines) / 44 t / ha (hardwoods)	Not reported
Hyypä <i>et al.</i> (2000)	High resolution optical	Air photos	Stem volume	DN values, ratios, NDVI-like transformations, squared sum of all channels	Boreal	0.48	73	46
Shugart <i>et al.</i> (2000)	High resolution optical	Air photos / high resolution satellite reconnaissance imagery	Aboveground biomass	Lag distance and maximum variance from semivariograms	Boreal	0.94	20 t / ha	17

Study	Sensor type	Sensor name	Dependent variable	Independent variables	Forest type	r ²	RMSE	RMSE (% of average)
Lefsky et al. (2001)	High resolution optical	ADAR	Aboveground biomass	DN values, texture (using the absolute difference technique), classifying the data and then calculating the fraction of each class in a larger pixel size	Natural Douglas-fir / western hemlock	0.47 (adjusted)	205 t / ha	30
Hyypä et al. (2000)	High resolution optical	SPOT Pan	Stem volume	Intensity	Boreal	0.35	85	54
Franklin (1986)	Medium resolution multispectral	Landsat TM simulator	Leaf biomass	Red DN values, vegetation index	Conifer, hardwood, mixed, chaparral	0.67	Not reported	Not reported
Fransson et al. (2001)	Medium resolution multispectral	SPOT XS	Stem volume	DN values, squared DN values, ratios	Boreal	0.86	31 m ³ /ha	24
Hyypä et al. (2000)	Medium resolution multispectral	Landsat TM	Stem volume	DN values, ratios, NDVI-like transformations, band differences, squared sum of channels, first three principal components.	Boreal	0.31	88 m ³ / ha	56
Hyypä et al. (2000)	Medium resolution multispectral	SPOT XS	Stem volume	DN values, NDVI-like ratios, band differences, squared sum of channels, channel ratios, first three principal components.	Boreal	0.44	79 m ³ / ha	50
Steininger (2001)	Medium resolution multispectral	Landsat TM	Aboveground biomass	DN values	Tropical secondary forest	0.70	Not reported	Not reported

Study	Sensor type	Sensor name	Dependent variable	Independent variables	Forest type	r ²	RMSE	RMSE (% of average)
Lefsky et al. (2001)	Medium resolution multispectral (single date)	Landsat TM	Aboveground biomass	Tasseled cap values	Natural Douglas-fir / western hemlock	0.31 (adjusted)	320 t / ha	47
Lefsky et al. (2001)	Medium resolution multispectral (multiple dates)	Landsat TM	Aboveground biomass	Tasseled cap values	Natural Douglas-fir / western hemlock	0.60 (adjusted)	239 t / ha	35
Hyypä et al. (2000)	Medium resolution hyperspectral	AISA	Stem volume	Intensity of thirty channels, principal components, ratios of principal components, NDVI-like ratios, band differences, channel ratios	Boreal	0.55	71 m ³ / ha	45
Lefsky et al. (2001)	Medium resolution hyperspectral	AVIRIS	Aboveground biomass	First 20 principal components	Natural Douglas-fir / western hemlock	0.38 (adjusted)	300 t / ha	44
Meeuwig <i>et al.</i> (1979)	High resolution optical	Air photos (type / scale not reported)	Total aboveground biomass per unit crown area	Crown width	Singleleaf pinyon pine, Utah juniper	Not reported	Not reported	Not reported
Alemdag (1986)	High resolution optical	Air photos (panchromatic, 1:1150)	Stem biomass	Crown area, tree height	Trembling aspen, white birch	0.98 (aspen), 0.56 (birch)	17 kg (aspen), 21 kg (birch)	Not reported

Study	Sensor type	Sensor name	Dependent variable	Independent variables	Forest type	r ²	RMSE	RMSE (% of average)
Alemdag (1986)	High resolution optical	Air photos (panchromatic, 1:1150)	Crown biomass	Crown area, tree height	Trembling aspen, white birch	0.79 (aspen), 0.47 (birch)	66 (aspen), 44 (birch)	Not reported
Alemdag (1986)	High resolution optical	Air photos (panchromatic, 1:1150)	Total aboveground biomass	Crown area, tree height	Trembling aspen, white birch	0.97 (aspen), 0.59 (birch)	19 kg (birch), 22 kg (aspen)	7 (aspen), 10 (birch)
Johnson <i>et al.</i> (1989)	High resolution optical	Air photos (type / scale not reported)	Crown mass	Crown width, tree height	Lodgepole pine, white spruce	0.85 (pine), 0.87 (spruce)	24 kg (pine), 63 kg (spruce)	Not reported

al. (1995) was successful in a natural temperate forest, but the authors found that they had to perform a classification of the imagery prior to creating regression equations to predict basal area, average tree height and crown biomass. The bole biomass was then calculated using the basal area and crown biomass as independent variables.

In general, researchers have found that the use of longer wavelengths (the L and P bands) generally produced better results than shorter wavelengths (C and X bands) Dobson *et al.*, 1992; (Le Toan *et al.*, 1992; Isrealsson *et al.*, 1994). This is most likely due to the ability of the longer wavelengths to penetrate more deeply into the canopy. Researchers have also found that crosspolarized data produced better results than data that are sent and received with the same polarization (Hussin *et al.*, 1991; Le Tuon *et al.*, 1992).

Another wavelength-dependent factor is the point at which the relationship between biomass and backscatter saturates. Imhoff (1995) found that the saturation point is ~100 t / ha using P band data, ~40 t / ha using L band data and ~20 t / ha using C band data. With these saturation points, Imhoff (1995) determined that a P, L and C band sensors were capable of predicting the biomass for 62, 37 and 25% of the world's forests, respectively.

Unfortunately, all current satellite-borne radar sensors are C band, and use a single polarization to send and receive electromagnetic energy. Recently, researchers have begun to use interferometry, a procedure that involves examining the coherence of data collected over a short time increment by two identical instruments. Currently this technique is possible using the ERS-1 and ERS-2 sensors, as well as several commercial airborne sensors (Li *et al.*, 2002). Hyypä *et al.* (2000) found that this technique produces more reliable results than the traditional single-image approach, and Fransson *et al.* (2001) demonstrated that it has a relatively high saturation point. Pulliainen *et al.* (2003), however, found that the accuracy of this technique was highly dependent on site conditions, such as wind speed, moisture, and whether the temperature was below freezing, all of which can change rapidly. They also found that using imagery from more than one date can increase the estimation accuracies, but only if the images were obtained under favorable conditions. In practice, this technique has been used with moderate success in the boreal forest (Hyypä *et al.*, 2000; Fransson *et al.*, 2001; Pulliainen *et al.*, 2003).

1.3.2 Estimation of biomass using statistically-processed passive optical remote sensing data

Several researchers have attempted to determine obtain forest biomass using statistically-processed (i.e., processed using a regression model) passive optical remote sensing data (see Table 2). These studies generally follow the following basic steps:

1. Determine the biomass in an area corresponding to either an image pixel or a group of pixels.
2. Extract the digital number (DN) values and / or transformed DN values for the pixel or group of pixels.
3. Use regression equations to relate the DN value and / or transformed DN values to biomass. Often multiple values are used as independent variables.

In two of the studies (Hyypä *et al.*, 2000; Lefsky *et al.*, 2001) the authors compared multiple data types. From these comparisons, it appears that for single date imagery, high spatial resolution equates to more accurate biomass estimates. Interestingly, Lefsky *et al.* (2001) found that the use of medium resolution data collected at multiple times of the year gave higher accuracies than results obtained using a single date images, including those with high spatial resolutions. The authors also found little benefit in using high spectral resolution data, although this result is questionable because they used a very simplistic processing technique that involved creating a regression equation using the first twenty principal components.

1.3.3 Estimation of biomass from high resolution passive optical imagery using object-oriented approaches

When high resolution passive optical data are available, an object-oriented approach can be used rather than statistical image processing. In this approach, regions of an image corresponding to objects of interest to the user are processed, rather than individual pixels (Geneletti and Gorte, 2003). For determining biomass, typically the object of interest is either a stand or an individual tree. Once the stands and trees have been identified, parameters such as crown width, tree height, and crown closure can be obtained manually or automatically using a

computer program. Stand or individual tree biomass can then be estimated using these parameters values. Volume estimation, which is very similar to biomass estimation, has been conducted in this way since the 1920s, and in many cases the results have been very good (Spurr, 1960, p. 382 - 392 and references therein).

Typically the volume estimation equations used in studies where the images were manually interpreted follow a number of standard forms. Of these, the two variable form is most common (Spurr, 1960, p. 385 - 387).

Forms:

One variable: $\text{Volume} = f(\text{mean stand height})$ or $\text{Volume} = f(\text{mean crown diameter})$

Two variable: $\text{Volume} = f(\text{mean stand height, crown closure})$

Three variable: $\text{Volume} = f(\text{mean stand height, crown closure, mean crown diameter})$
or $\text{Volume} = f(\text{mean stand height, crown closure, trees per unit area})$

In several relatively recent studies, researchers have used manual versions of this technique to estimate biomass instead of volume. Meeuwig *et al.* (1979) used crown widths to estimate the whole-tree biomass of singleleaf pinyon pine and Utah juniper trees from air photos. Alemdag (1986) determined stem, crown and whole-tree biomass for trembling aspen and white birch using crown diameter and height measurements, and Johnson *et al.* (1989) successfully determined the crown biomass of lodgepole pine and white spruce using photo-derived crown widths and tree heights.

During the past decade, attention has focused on developing computer based techniques for identifying trees and stand boundaries, and for obtaining parameters for these objects (e.g., Pollock, 1994; Gougeon, 1995; Dralle and Redemo, 1996; Brandtberg and Walter, 1999; Culvenor *et al.*, 1999; Larsen, 1999; Wulder *et al.*, 2000). These techniques will be discussed extensively in Section 2. Unfortunately, none of these publicly described techniques provide biomass or volume as an output. The Forest Assessment and Classification Tool (Falcon

Informatics, Doylestown, PA) is able to determine stand volume based solely on passive optical data, but it is a proprietary, in house system and its accuracy is not known.

1.3.4 Light detection and ranging (LiDAR) estimation of biomass

Light detection and ranging (LiDAR) is a relatively new type of remote sensing that promises to provide biomass estimation accuracies that equal or exceed those obtained using other remote sensing techniques. LiDAR works by emitting a pulse of visible or near-infrared laser light, and measuring the time it takes for the pulse to return to the sensor. In some cases, multiple returns from the laser pulse can be measured, and some instruments are able to determine the intensity of the return (Dubayah and Drake, 2000).

Although many types of LiDAR systems exist, they all fall into two basic categories: Large footprint and small footprint. In large footprint systems, the laser pulse covers a relatively large area (typically hundreds of square meters), and generally the intensity of the return is measured at frequent intervals. To date all large footprint systems are experimental devices constructed by research institutions. Small footprint systems use much more narrowly focused laser beams (typically covering less than 0.25 m²). Although small footprint instruments have recently become available that record intensity at regular intervals through the canopy, generally intensity is not recorded. Instead these instruments typically measure the timing of the start and end of the returned signal (Dubayah and Drake, 2000; Hyypä and Inkinen, 2002).

Due to the fundamental differences between large and small footprint LiDAR, different processing techniques are used for the two data types. To process large footprint data, past studies have derived metrics from the LiDAR data, and then used regression analysis to create equations relating these metrics to forest stand properties (see Table 2). A large number of metrics have been proposed and used in these studies. Lefsky *et al.* (1999b) and Lefsky *et al.* (2002) used four metrics related to canopy height (maximum canopy height, median canopy height, mean canopy height, and quadratic mean canopy height). Drake *et al.* (2002) also used height metrics, but additionally used metrics related to return intensity (canopy height, height of medium energy, the ratio of canopy height to the height of median energy, and the fraction of energy that was returned by the ground). Means *et al.* (1999) examined seven metrics related to

canopy height, ground elevation, and return intensity. They found that biomass could be accurately predicted based only on the mean canopy height, but could be improved by including the quadratic mean canopy height and the canopy reflectance sum. Lefsky *et al.* (1999a) increased the number of metrics to twenty by adding metrics obtained using the canopy volume method (CVM). CVM metrics are calculated by assigning each point in a returned energy spectrum to a class. This class is based on the proportion of the total energy that was returned before a point of interest in the spectrum, and the proportion of the total energy that was returned after the point of interest. A 5x5 grid of returns is examined, and the average height for each class is calculated. Although the authors created twenty metrics, only two were used in the final biomass prediction equation.

Large footprint LiDAR data have been shown to give excellent biomass estimates in a deciduous forest in the eastern US (Lefsky *et al.*, 1999a; Lefsky *et al.*, 2002), conifer forests in the northwestern US (Lefsky *et al.*, 1999b; Means *et al.*, 1999; Lefsky *et al.*, 2002), a tropical forest in Costa Rica (Drake *et al.*, 2002), and a Boreal forest in Canada (Lefsky *et al.*, 2002). Lefsky *et al.*'s (2002) work showed that it is possible to create a single relationship between LiDAR-derived metrics and biomass for multiple forest types.

Small footprint LiDAR data are very different from large footprint data, and some authors (Means *et al.*, 1999; Dubayah and Drake, 2000; Drake *et al.*, 2002) have argued that this type of data is poorly suited for forest inventories. This is because 1) there is no assurance that a laser pulse will strike the tops of the trees, 2) it is unclear whether a laser pulse has struck the ground, and 3) extensive flying is required to obtain adequate data for analysis. Despite these criticisms, studies using small footprint LiDAR to estimate biomass (Nelson *et al.*, 1988; Naesset, 1997; Hyppa and Inkinen, 2002; Popescu and Wynne, 2002; Popescu *et al.*, 2003; Popescu and Wynne, in press) have yielded excellent results (see Table 2). Small footprint LiDAR also has the advantage of being available commercially from a number of companies. This makes it suitable for routine use. Unfortunately, unlike large footprint LiDAR, methods available for processing these data do not follow similar approaches, and it is not yet clear which method is best for a given scale and type of forest.

One of the earliest attempts to use small footprint LiDAR to predict biomass was Nelson *et al.*'s (1988) attempt to use a laser profiling system to determine the biomass of forest tracts in Maryland. To do this, the authors extracted six height variables from the data, and used regression analysis to predict the tract biomass. This technique yielded high accuracies, and it was determined that it was unnecessary to stratify trees by species.

Naesset (1997) used a scanning LiDAR system to determine the biomass of softwood stands in southern Norway. The author's technique was to extract three variables from the LiDAR data: Mean stand height, mean height of all laser pulses within the stand, and mean canopy cover density. Regression analysis was then used to create a relationship between these variables and stand volume.

Hyypä and Inkinen (2002), Popescu and Wynne (2002), Popescu *et al.* (2003), and Popescu and Wynne (in press) have recently developed object-oriented algorithms that attempt to identify individual trees and measure their properties directly. These algorithms will be discussed in detail in Section 2.

1.3.5 Summary of techniques used for remotely estimating stand or tree biomass

Data from radar, passive optical, and LiDAR sensors have all been used by researchers to determine stand or tree biomass. Radar data traditionally has been processed by developing equations relating backscatter at one or more frequencies and polarizations to stand biomass. However, the equations that result from this approach saturate at high biomass levels, making this approach ineffective for many areas (Imhoff, 1995). Interferometric radar is less susceptible to saturation, but it has not been extensively tested so it is not known whether good results can be obtained in all forest types.

Passive optical data with a wide range of spatial and spectral properties have been used to determine biomass. Typically one of two approaches are used: Either the biomass is predicted statistically using the DN values of pixels or groups of pixels, or an object-oriented approach is used in which properties of individual trees or stands are obtained from the imagery and used to predict biomass. For the first approach, researchers have found that results improve when either the spatial or temporal resolution is increased (Hyypä *et al.*, 2000; Lefsky *et al.*, 2001). Object-

oriented approaches have been performed manually for years using aerial photographs (Spurr, 1960, p. 382 - 392 and references therein) and it is likely that it is still the most common means of obtaining biomass and volume estimates from remotely sensed imagery. Currently only one program is able to use an object-oriented approach to obtain biomass from aerial photographs, and it is proprietary and in house.

LiDAR remote sensing is a promising new technique for determining biomass. Currently two types of LiDAR instruments are in use: Large footprint LiDAR in which the return from a pulse of light several hundred square meters in area is measured, and small footprint LiDAR in which the pulse is typically less than 0.25 m² in area. Large footprint imagery is typically processed by extracting metrics from the data and developing regression equations relating the metrics to biomass. These data have been used very successfully in a wide range of forest types. Small footprint LiDAR can be processed either by statistically relating values obtained from the data to biomass or using an object-oriented approach. Both techniques have been used successfully, although small footprint-based approaches have been tested in fewer forest types than large footprint based approaches have.

1.3.6 Justification for using aerial photography and small footprint LiDAR data and an object-oriented approach

Previous research has shown that accurate biomass information can be obtained using multiple sensor types and both statistical and object-oriented data processing. However, there are benefits and disadvantages associated with each data type, and with both processing approaches. In this study, an object-oriented approach was used to process aerial photograph and small footprint LiDAR data. To justify this decision, the advantages and disadvantages of these data and this processing technique will be identified and weighed.

1.3.6.1 Advantages and disadvantages of high spatial resolution passive optical and small footprint LiDAR imagery

Advantages

1. Aerial photographs and other high resolution passive optical imagery are readily available for most areas.

High resolution optical data in the form of air photos have been routinely collected by the public and private sectors since the end of World War I (Lillesand *et al.*, 2004, p. 61), and there are currently companies in many countries and in all regions of the United States that specialize in air photo collection and processing. Many of these companies (e.g., Earthdata International, Frederick, MD and Spectrum Mapping, Easton, MD) are also able to provide small footprint LiDAR data. These data are supplemented by high resolution satellite imagery from the QuickBird (DigitalGlobe, Longmont, CO) and Ikonos (Space Imaging, Inc., Thornton, CO) satellites, which provide regular coverage of nearly the entire earth.

2. Small footprint LiDAR imagery and high resolution passive optical imagery can provide information on what lies below the canopy

Most researchers who wished to obtain subcanopy information have used large footprint LiDAR (Dubayah and Drake, 2000) or radar data (e.g., Proisy *et al.*, 2002). However a study by Todd *et al.* (2003) found that with proper processing, small footprint LiDAR imagery could provide this information. Additionally, research by Treitz (2001) suggests that variogram analysis performed on high resolution passive optical imagery is able to detect subcanopy features which arise due to multiscattering. This effect is particularly pronounced in near infrared imagery.

3. These data types enable individual tree based algorithms to be used

Object-oriented algorithms are based on the properties of multiple connected pixels that represent an object (Geneletti and Gorte, 2003). Therefore it logically follows that in order to use an object-oriented algorithm, the pixel size must be smaller than the object that will be identified and analyzed. In the case of tree crowns, the area of a tree crown is less than that provided by commonly available types of medium resolution imagery. For example, Bragg (2001) measured the crown widths of 1,613 trees in Wisconsin and Michigan belonging to 24 species, and found that the average crown width was 5.3 m, and the average maximum crown width for the species was 10.6m. Although the average and maximum crown widths may be larger in other forest types (e.g., tropical rainforests), these values suggest that commonly used medium resolution satellite imagery such as Landsat TM with a 30m pixel size (Lillesand *et al.*, 2004, p. 406) would be unsuitable for the task. Similarly, large footprint LiDAR systems are typically configured to have a beam width equal to or greater than the average crown diameter (Dubayah and Drake, 2000). By contrast, high resolution passive optical data from satellites such as Ikonos and Quickbird, as well as many orthophotographs (such as the ones used in this study) have a resolution that is finer than 5.3 m. Additionally, it is possible to generate canopy height models (CHMs) with very fine pixel sizes (e.g., 50 cm) using small footprint LiDAR data.

4. For passive optical data, statistical processing approaches may benefit from higher resolution

In two comprehensive studies (Hyypä *et al.*, 2000; Lefsky *et al.*, 2001), researchers examined the accuracy of biomass retrievals made using statistically processed medium and high spatial resolution imagery. Both studies found that using high spatial resolution imagery resulted in higher prediction accuracies.

Disadvantages

1. High resolution passive and active optical data are expensive

High resolution passive optical and small footprint LiDAR data are more expensive to collect than medium resolution optical data and radar data collected from spaceborne systems (Table 1).

2. These data types are storage intensive and are time consuming to process

Whenever the pixel size is decreased by 50%, there is a 400% increase in the number of pixels needed to cover a corresponding area. For example, to cover the area represented by just one 30m Landsat TM pixel, 3600 pixels in a 50 cm LiDAR CHM must be used. For this reason, using high spatial resolution imagery often requires the user to store large amounts of data, even for small areas. Similarly, processing high spatial resolution imagery is generally more time consuming than processing lower resolution imagery due to the much greater number of pixels that must be processed.

3. Some features may be sacrificed in order to achieve the high spatial resolution or narrow beam width

When high spatial resolution passive optical imagery and small footprint LiDAR imagery are collected, compromises must be made due to engineering considerations. In passive optical imagery, this means that energy at a wide range of wavelengths generally must be averaged in order to achieve a useful signal to noise ratio (SNR), thereby precluding the collection of data that has both high spectral and spatial resolution. For LiDAR imagery, small footprint LiDAR measurements require that a pulsed rather than a continuous beam be used (Wehr and Lohr, 1999). As a result, less light per measurement interacts with the surface, and each measurement

must be processed more quickly. For this reason the return intensity of most small footprint systems is not fully digitized like it is with large footprint systems.

4. Obtaining historical data may be problematic

LiDAR data was first collected in the late 1970s (Lillesand *et al.*, 2004, p. 725), and until very recently most data collection efforts have been very small scale. This makes these data unsuitable for many historical studies. Aerial photographs have been routinely collected since the end of World War I (Lillesand *et al.*, 2004, p. 61). However, the coverage of these data is irregular, and frequently different scales and film types were used. Additionally, many of these images are difficult to obtain because they were collected by a wide variety of public agencies and private companies. This makes obtaining and using high resolution aerial photographs difficult. Although high spatial resolution satellite imagery will solve these problems for future researchers, it only became available in 1999 (Soliday, 2000). The availability of medium resolution optical data and radar data is much greater, since archives of historical data with consistent properties exist for much of the earth. These archives exist from 1972 to the present for medium resolution optical data, and 1991 to the present for radar data (Lillesand *et al.*, 2004; 404, 702).

5. The additional data provided by high spatial resolution passive optical imagery may confuse some algorithms

Most of the commonly used approaches for classifying remotely sensed images are based on the information contained in individual pixels (see Lillesand *et al.*, 2004, p. 554 - 586). This approach may be unsuccessful in high spatial resolution data if single objects are represented by multiple pixels. For example, a healthy tree might consist of pixels corresponding to bark, leaves, and shadows, all of which have very different spectral characteristics. A way would need to be found to classify all of these materials as 'healthy tree,' which may lead to unsatisfactory results if another class consists of the same materials but in different proportions (e.g., a 'defoliated tree' class).

1.3.6.2 Comparison between object-oriented and non-object-oriented approaches

In this study, forest and tree biomass values were determined using an object-oriented approach that was applied to high resolution aerial photograph and small footprint LiDAR imagery. Using an object-oriented approach has a number of advantages and disadvantages compared to statistically-based image processing techniques.

Advantages

1. There is no boundary effect

In many areas, a significant fraction of forests lie along edges. This is true even for rural areas. For example, unpublished work performed by Bortolot in rural Louisa County, Virginia, showed that approximately 13% of the forest lies within 31m of a forest edge, and 32% lies within 91m of a forest edge. For techniques based on medium resolution imagery or based on textural windows, this will make developing statistical relationships challenging because it is difficult to adjust for the effects of non-forested areas that are included in the pixels. Edge areas are not a problem for tree based approaches, unless an algorithm is used that has difficulty separating tree crowns from certain non-tree land cover classes. Therefore, an investigator may want to choose a tree based method for areas with highly fragmented forests, such as those covering most of the eastern US.

2. The technique is relatively immune to differences in illumination, atmospheric conditions, and phenology.

In many analyses based on statistical approaches, differences in illumination, atmospheric conditions, and phenology within a scene or among multiple scenes are problematic. This is because these factors affect the DN values in the image (Lillesand *et al.*, 2004, p. 37, 247).

With object-oriented approaches, this is less of a problem since patterns of DN values are examined, rather than the DN values of single pixels. These patterns tend to be relatively immune to differences in illumination, the atmosphere, and some phenological changes (e.g., the difference between late spring and late summer in a hardwood stand). This is a major advantage when it is necessary to process images acquired in different areas or on different dates.

3. It is possible to use the same equations in a variety of stand types

Although some authors have developed biomass and volume equations for individual species (e.g., Meeuwig *et al.*, 1979; Alemdag, 1986), research by Avery (1958) showed that it is possible to successfully use a single equations to predict volume (and therefore most likely biomass as well) for a variety of hardwood and softwood species using parameters obtained from aerial photographs. With the exception of Lefsky *et al.* (2002), past studies using statistical image processing approaches have developed models for specific forest types, and it is unclear whether the equations could be transferred to new areas.

Disadvantages

1. Object-oriented approaches are generally slower than statistical approaches

Algorithms that are based on statistical manipulations of values extracted from medium resolution optical or radar data are generally very quick once the equations are derived, typically on the order of a few seconds even for a large image. Using an object-oriented approach is often more time consuming, especially if the extraction is performed manually. For example, in an image of eleven to seventeen year old loblolly pine plantations (false color, spatial resolution =50 cm), manually locating the trees in a 0.07 ha plot and measuring their crown diameters takes approximately thirty minutes. Automated techniques are somewhat quicker, but may require time to set parameters. The automated technique described in Section 2 of this dissertation is able to process a 0.07 ha photoplot in approximately five seconds, but setting the parameters takes

several hours of computer time using a computer with a 2.4 GHz Intel Xeon processor and sufficient memory to store and process the images in RAM.

2. The pixels must be smaller than the object of interest

As discussed in Section 1.3.6.1, object-oriented approaches require that the pixels be smaller than the object under examination. This means that high spatial resolution imagery may be required for some objects. For example, an object-oriented approach could not be used on Landsat TM imagery if the objects of interest are individual trees.

1.3.6.3 Weighing the advantages and disadvantages

There are both significant advantages and disadvantages to using high spatial resolution passive optical and LiDAR data and an object-oriented approach. The key advantages that pertain to determining forest biomass and density are 1) the immunity to boundary effects, 2) the ability to work in multiple stand types, and 3) that minor scene-to-scene differences in illumination, the atmosphere, and phenology should have relatively little effect. These advantages are very important for many applications. Boundary effects are perhaps the most important, since as shown by the Louisa County, Virginia example, large portions of many forests lie close to roads, agricultural fields, and other non-forest land classes. Using medium resolution imagery or a statistical approach is likely to lead to large errors in these situations due to problems with mixed pixels and the spectral similarity of forests and some non-forest land cover classes such as agricultural fields. Although many commercial forestry companies are only interested in a single type of forest (e.g., loblolly pine plantations), government programs such as FIA need to inventory a wide variety of forest types. A program that would work different forest types without the need to develop new equations would be highly advantageous for applications like this. Finally, for studies covering large areas or spanning multiple years, having some immunity to illumination, atmospheric, and phenological differences would be very important, since these effects are likely to be present in these situations.

The most important shortfalls of using an object-oriented approach and high spatial resolution imagery are the need for extensive computer resources, the cost of the data, and the

lack of historical imagery. Using high spatial resolution imagery consumes a lot of disk space, and high resolution data and the use of an object-oriented approach can make processing images a very slow process. However, the process is still faster than conducting a ground based survey, and future advances in computers are likely to reduce the importance of this issue. The expense of these data may limit the application of this approach to studies that are either small scale or very well funded, but many useful studies and study areas (e.g., the two BOREAS study areas) fall into one or both of these categories. The lack of historical imagery may preclude the use of this technique in change detection studies. However, generally only the current properties of the forest are of interest, so this is a relatively minor issue. Additionally, in the future this problem will decrease since archives of high spatial resolution satellite imagery are currently being assembled by DigitalGlobe (Longmount, CO) and Space Imaging (Thornton, CO).

Overall, it appears that an object-oriented approach combined with high spatial resolution imagery has several advantages over medium resolution imagery and statistical data processing. The disadvantages associated with the approach relate to current limitations in computer technology, and the cost and availability of appropriate imagery. However, these are likely to become less problematic in the future, and are currently less of a problem for smaller scale studies.

1.4 Summary and the objectives of the dissertation

Since the 1850s, atmospheric CO₂ levels have increased greatly due to human activity. This increase is highly problematic because it impacts many of earth's systems in ways that may reduce the quality of life the earth can provide for its inhabitants. Forests have been proposed as a means of reducing CO₂ levels, because trees absorb CO₂ during photosynthesis and store it as biomass. In order to quantify the amount of CO₂ absorbed by different forest types, it is important to obtain accurate estimates of forest biomass. Remote sensing techniques provide a means of accurately estimating forest biomass that is inexpensive and rapid in comparison to ground based surveys. Several remote sensing techniques have been shown to be successful in biomass prediction studies, and the choice of technique depends on the scale of the study and the resources that are available.

The major objective of this dissertation is to engineer a new object-oriented approach to measuring trees in remotely sensed images, and to write a program based on this approach. The approach locates trees in normal and false color digital aerial photographs and small footprint LiDAR imagery of different forest types that have a range of spatial resolutions, and measures the crown diameters and heights (LiDAR imagery only) of the trees in the images. These data are then used to determine stand biomass. The novel aspect of the approach is that it relies on several generic tree attributes, and uses training data to determine how these generic attributes are used. Using training data in digital image processing is a common and accepted practice in remote sensing, and is an integral part of popular techniques such as Gaussian maximum likelihood classification. However, this is the first study to use training data in conjunction with an individual tree-based algorithm, which should enable it to be more flexible than existing algorithms. It is also the first to work with both orthophotographs and LiDAR data. Section 3.2 discusses the approach that was developed to meet this objective.

A secondary objective was to perform a first test of the program to determine whether the new approach is viable and merits further study, and to examine factors that affect the results it obtains. This objective was met by:

1. Quantifying the accuracies of the density and biomass estimates obtained by the program using digital orthophotographs and LiDAR data by comparing them with data obtained in a field survey or by a human photointerpreter.
2. Examining the effect of training sample size on accuracy.
3. Comparing six methods for combining multiple computer estimates of density or biomass into a single estimate.
4. Determining whether the trees located by the computer program correspond to the trees located by a human photointerpreter.
5. Comparing results obtained using the modified Nelder-Mead simplex optimization algorithm to those obtained using a genetic optimization algorithm.

These five components of the objective and the methods for addressing them are discussed in Section 3.6. Results related to these components are given and discussed in Chapter 4.

Chapter 2

Literature review

2.1 Introduction

In this study, a new technique was developed for determining stand density and biomass using tree and stand attributes that are directly measured from aerial photographs or small footprint LiDAR images by a computer algorithm. High resolution aerial imagery was first employed for forest inventory in 1887, when a German forester used photographs collected from a balloon to produce a forest map (Spurr, 1960, p. 349). Since that time, high resolution imagery in the form of aerial photographs has become a mainstay of forest inventory in many parts of the world. Traditionally these data have been processed by human interpreters. Although human interpreters can detect more subtle patterns than current computer algorithms and make better use of contextual information, manual image processing is quite time consuming, and two photointerpreters interpreting the same image may produce markedly different results (Meyer and Worley, 1957). Recently, interest has developed in using object-oriented computer image processing techniques to replace or supplement human interpretation of aerial images, and several algorithms have been proposed for locating trees in these types of images and in some cases delineating their crowns and determining biomass. Additionally, high resolution aerial imagery no longer consists solely of passive optical images. As discussed in the introduction, small footprint LiDAR imagery has been used successfully by a number of studies to obtain forest parameters from the air.

2.2 Past methods for using object-oriented computer vision techniques for locating trees in high resolution imagery and delineating their crowns

Numerous researchers have developed techniques for locating trees in high resolution images and delineating the trees' crowns. Approaches used by these algorithms include valley-following, delineation using curved edge segments, template matching, peak-finding methods, and segmentation. These approaches are summarized below, and the results are compared in Section 2.1.2.

The valley following technique was developed by Gougeon (1995), and involves isolating individual trees by thresholding the image to separate trees from shade, and then using filters and a rule-based operator to separate trees from one another.

Brandtberg and Walter (1999) developed an algorithm that delineates individual tree crowns using curved edge segments. Their approach involves 1) using gradient maxima to locate edge segments in images that were smoothed by different amounts, 2) connecting the edge segments to form curve segments and eliminating small curve segments or ones that are concave, 3) finding circles that are tangential to the remaining curve segments, 4) finding the centers of the circles (evolutes), 5) merging the curve segments and evolutes found at different scales, and 6) growing trees starting at areas in which there was a high density of evolutes and constraining the growth based on the curve segments.

Template matching techniques involve the creation of synthetic images of individual trees, known as templates. The templates are passed over the image being processed, and locations in the image where the template closely matches the image receive a high score, whereas locations in which there is a poor match receive a low score. The result is an image showing the locations of trees in the original image that have a similar appearance to the template. This technique was developed by Pollock (1994), and the templates addressed crown shape, leaf distribution, sensor geometry and illumination. Larsen (1999) built on Pollock's work by refining the way in which the scattering and absorption of needles is modeled, and by modeling the reflectance of the ground plane.

Several authors have used the fact that the tops of trees appear to be brighter (passive optical) or higher (small footprint LiDAR) than the edges of trees, giving them a domed shape in

a three dimensional plot in which the X and Y axes of the plot correspond to the X and Y axes of the image, and the Z axis corresponds to the DN value at the X and Y position (see Figure 1).

Dralle and Rudimo (1996) took advantage of this phenomenon by passing a Gaussian smoothing filter over the image and then taking the threshold. The variance parameter for the Gaussian filter was set based on the known tree density.

Wulder *et al.* (2000) used an approach that is somewhat similar to the one used by Dralle and Rudimo (1996), but instead of using smoothing followed by thresholding they used local maximum filters. Local maximum filters obtain the maximum value within a window. The authors used both fixed window sizes and variable window sizes determined using one of two techniques: 1) Calculating the semivariance for each pixel in the image, and then using the range to determine the optimal window size at the point and 2) using slope breaks.

Culvenor *et al.* (1999) used a different local maximum approach. Their approach found local maxima by examining a user specified number of neighbors along the horizontal, vertical and diagonal axes. If the center pixel had a value that was higher than the average of the neighboring pixels, then it was considered to be a local maxima. The local maxima from all axes were combined to give an image showing the number of directions in which the pixels were local maxima, and this image was thresholded. A similar process was then used to produce an image showing the local minima, and the local minima image was processed to yield segments that were only one pixel thick. The local maxima image was refined to identify seed pixels, and these seed pixels were expanded outward until they either encountered a pixel that had a DN value greater than a user defined DN value or a local minima was encountered.

Popescu *et al.* (2003) and Popescu and Wynne (in press) used a local maximum approach for obtaining forest parameters from small footprint LiDAR data. This work involved the following steps:

1. Create a canopy height model (CHM).

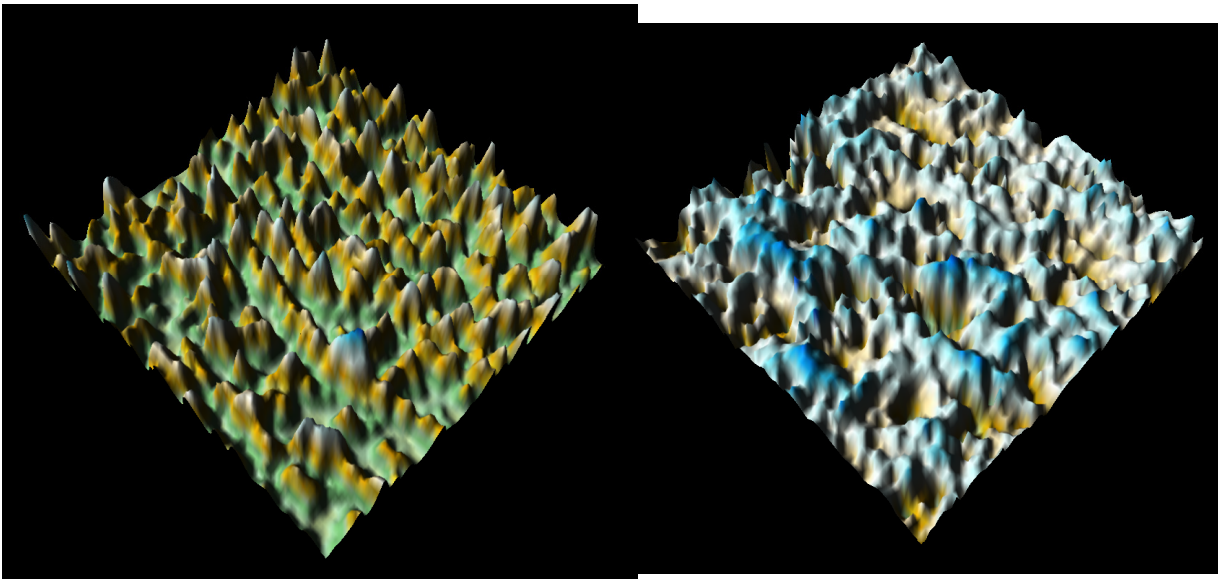


Figure 1. Three dimensional diagrams constructed from corresponding images of a pine plantation (Plot 1: see Section 3.4). In the diagram on the left, the X and Y axes correspond to the X and Y axes in the image, and the Z axis corresponds to the DN value in the near-infrared band. The original image had a spatial resolution of 50 cm. In the diagram on the right, the X and Y axes correspond to the X and Y axes in a small footprint LiDAR canopy height model, and the Z axis corresponds to the canopy height. The LiDAR points were interpolated to form a grid with a spatial resolution on 0.5 m. In both images, trees appear as concave down peaks, and are separated from adjacent trees by concave up troughs corresponding to shadows in the photograph or low areas in the LiDAR image.

2. Determine the heights and locations of individual trees by examining variable-sized regions surrounding each pixel in the CHM. The pixel is considered to correspond to the location and height of an individual tree if it is a local maximum.
3. Use the tree locations and stand areas from a GIS coverage to determine the stand density.
4. Determine the crown widths by extending two perpendicular lines from each tree center, fit a fourth-degree polynomial trend line, and locate the center of the trend line's trough.

5. Create regression equations to predict stand biomass using height, density, and crown width-based metrics.

This technique was shown to work well for hardwood and softwood stands in Central Virginia.

Hyypä and Inkinen (2002) also processed small footprint LiDAR data, but they used a segmentation-based approach on data from a boreal forest. Their technique involved using a watershed segmentation algorithm to delineate the boundaries of individual trees, and then looking at the maximum value within each segment and the dimensions of the segments to determine the height, diameter and volume the individual trees. This technique produced excellent results.

McCombs *et al.* (2003) applied a local maximum approach to small footprint LiDAR data, near infrared high spatial resolution passive optical data, and fused LiDAR and passive optical data. Prior to processing, the first return for the LiDAR data were interpolated to form a canopy surface model (CSM). The data were fused by examining a window in the passive optical data with a 3.048 m radius. For the center pixel, the percentile of the brightness of that pixel was determined, and multiplied by the height of the pixel in the CSM image to give a fused value.

Their approach for the three datasets was to examine an area within a 1.219 m radius of a center pixel, and to consider the center pixel to represent the top of a tree if the center pixel was the highest or brightest in the window. The window radius was set based on prior knowledge of the trees in the test images

2.3 Results of past studies

2.3.1 The valley following approach

Gougeon (1995)

Site: Uniform stands of red pine, red spruce, white spruce and Norway spruce.

Imagery: MEIS-II (airborne pushbroom scanner), 31cm ground resolution. Only a single

band was used.

Results: The number of trees were within 7.7% of ground counts. 71% of the trees identified were the same as those identified by a human interpreter.

2.3.2 Crown identification using curved edge segments

Brandtberg and Walter (1999)

Site: Naturally regenerated mixed and pure stands. Species present included Scots pine, Norway spruce, birch and aspen. The average stand age was 80 years old.

Imagery: Scanned 1:2,000 air photo, 10 cm resolution.

Results: 54% of the trees identified on the ground were properly identified.

2.3.3 Template matching

Pollock (1994)

Site: 587 dominant and co-dominant trees. Nine coniferous species, ten deciduous species.

Imagery: MEIS-II, 36 cm resolution.

Results: 57% of the trees identified by the algorithm corresponded to known tree locations. The identification accuracy was higher for conifers than for hardwoods.

Larsen (1999)

Site: 48 year old pure spruce stands.

Imagery: Scanned air photo, 15 cm resolution.

Results: An average of 95% of the individual trees were located.

2.3.4 Peak finding methods

Dralle and Rudemo (1996)

Site: 48 year old Norway spruce plantation with subplots subjected to different thinning treatments.

Imagery: Scanned 1:4,000 air photo, 15 cm resolution.

Results: The predicted density was 0.8 percent less than the actual density for the whole plot. The errors in the predicted densities for individual subplots ranged from -6.9 to 5.0%.

Wulder *et al.* (2000)

Site 1: Forty year old Douglas-fir and western red-cedar plantation.

Site 2: Old growth stand consisting primarily of Douglas-fir.

Imagery: MEIS-II, pixels resampled to 1m.

Results: See Table 3

Culvenor *et al.* (1999)

Site: Simulated trees with an average crown radius of 5 m with a standard deviation of 1.5m and a maximum overlap of 35%. 400 trees per hectare.

Imagery: Simulated 1m with various view and sun angle parameters.

Results: Up to 92% of the crowns were located.

Table 3. Results obtained by Wulder *et al.* (2000) using panchromatic data and either fixed window sizes or variable window sizes calculated using the semivariance (SVR) or slope break (SB) technique. For the fixed window sizes, the window sizes are given in pixels. The values indicate the proportion of trees in each category.

	Fixed (3x3)	Fixed (5x5)	Fixed (7x7)	SVR	SB
Combined					
Correct	0.67	0.50	0.30	0.64	0.62
False positive	0.22	0.05	0.02	0.19	0.11
Missed	0.33	0.50	0.70	0.36	0.38
Plantation					
Correct	0.62	0.43	0.21	0.60	0.56
False positive	0.05	0.02	0.01	0.05	0.03
Missed	0.38	0.57	0.79	0.40	0.44
Mature					
Correct	0.80	0.72	0.60	0.76	0.80
False positive	0.78	0.10	0.08	0.64	0.38
Missed	0.20	0.28	0.40	0.24	0.20

Popescu *et al.* (2003), Popescu and Wynne (in press)

Site: The Appomattox-Buckingham State Forest in Central Virginia. Pure hardwood, pure softwood and mixed stands were analyzed. Very small (0.017 ha) plots were used in this study.

Imagery: Small footprint LiDAR (Aeroscan) interpolated to form a canopy height model with a 50 cm spatial resolution.

Results:

Stand density:

The estimated stand densities had r^2 values ranging between 0 and 0.2626, depending on the species, the range of tree diameters considered, and independent variables used in the density prediction equation.

Crown width:

For dominant trees, the crown width was predicted with an r^2 between 0.62 and 0.63 for both hardwoods and softwoods.

Biomass:

For softwoods, the r^2 was 0.82 (RMSE = 29 Mg / ha). For hardwoods, the r^2 was 0.32 (RMSE = 44 Mg / ha).

Height:

The r^2 for mean subplot heights ranged from 0.35 to 0.85 depending on whether all trees were included in calculating the mean, or just the ones with the greatest height and stem diameter.

McCombs *et al.* (2003)

Site: 15-year old loblolly pine spacing trial in Mississippi. Trees were planted at a 2.4 x 2.4 m or a 3.0 x 3.0 m spacing.

Imagery: Small footprint LiDAR imagery (Airborne Lidar Topographic Mapping System) interpolated to form a canopy surface model with a 0.1524 m pixel. Digital passive optical data (Spectral Visions) collected in four bands with a spatial resolution of 0.61 m. Only the near infrared band was used. Fused LiDAR and passive optical data.

Results: For the trees planted at the 2.4 x 2.4 m spacing, 64.67% of the trees were correctly identified using the LiDAR data, 78.61% were correctly identified using the passive optical data, and 83.50% were correctly identified using the fused data.

87.27, 92.37, and 94.84% of the trees were identified in the stands with the 3.0 x 3.0 m spacing using LiDAR, passive optical, and fused data respectively.

2.3.5 Segmentation method

Hyypä and Inkinen (2002)

Site: Natural boreal forest consisting primarily of Norway spruce and Scots pine.

Imagery: Small footprint LiDAR (TopoSys-1) interpolated to form a canopy height model.

Results:

Height: For individual trees, the authors obtained an r^2 of 0.969, and an RMSE of 0.98 m. For mean stand height, the r^2 was 0.8341, and the RMSE was 2.3 m (9.6%)

Volume: The authors obtained an r^2 of 0.98, with an RMSE of 16.5 m³ / ha (9.5%)

Basal area: An r^2 of 0.89 was obtained with an RMSE of 16.5 m³ / ha (9.5%)

2.4 Comparison and criticism of these results

Based on the results seen in Section 2.1.2, it appears that all of these techniques were successful. However, it is very difficult to directly compare the results since no standard test dataset or evaluation technique exists, and most of the programs are not freely available for independent evaluation and comparison. Additionally, all of these techniques are limited in their design because they were constructed for a single type and scale of imagery, and a single type of forest. Changing the programs to work with additional image and photo types would generally require expert image processing and programming knowledge which would exceed the capabilities of many potential users.

Chapter 3

Methods

3.1 Overview

There were two objectives for this study. The primary objective was to develop an algorithm for using an object-oriented approach to locate and measure trees in orthophotographs and small footprint LiDAR CHMs, and then to write a program that implements this algorithm. The secondary objective was to perform a first test of the algorithm. Sections 3.2 and 3.3 of this chapter describes the algorithm and explains how it was developed, and Sections 3.4 to 3.6 provide details on how the algorithm was tested.

3.2 Image processing algorithm

3.2.1 Generic attributes

The tree finding procedure was designed by examining the characteristics human interpreters use to identify and delineate trees in images. Figure 2 shows sections of a false color orthophotograph of a pine plantation and a LiDAR CHM. In the photograph, it can be seen that 1) the tree crowns are approximately round in shape, 2) leaf-on trees are pink or red in false color photographs, and although not shown, they are green in normal color orthophotographs, and 3) trees are often surrounded by either dark shadows or (not shown) bare or sparsely vegetated soil. In the LiDAR canopy height model, the trees are also round in shape. Although color cannot be used to distinguish trees from their surroundings, trees in LiDAR images are higher than their surroundings.

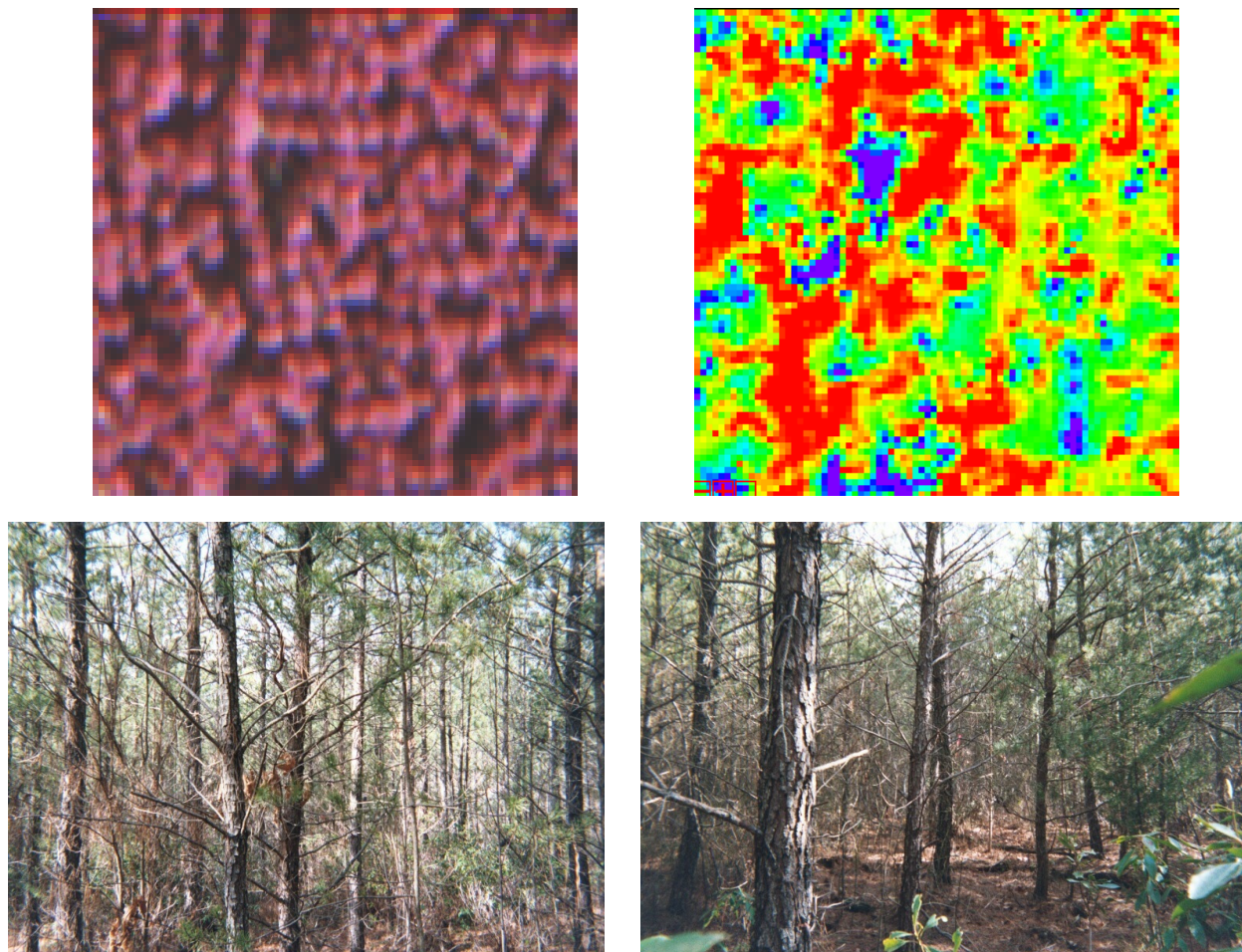


Figure 2. Orthophotograph and LiDAR images of a pine plantation and corresponding ground photographs. The top left image shows a false color orthophotograph with a spatial resolution of 50 cm acquired over a loblolly pine plantation (plot 1: see Section 3.4) in the Appomattox-Buckingham State Forest. This image illustrates three properties of trees in aerial photographs: 1) They are approximately round in shape, 2) in false color images they are pink or red in color, and 3) they may be surrounded by dark shadows. The top right image shows a LiDAR CHM of the same area, with an interpolated spatial resolution of 50 cm. In this image, warmer colors correspond to greater heights. The LiDAR image shows trees in a LiDAR canopy height model are relatively round in shape, and tree tops tend to have a relatively high height and are surrounded by areas of low height. It is notable that the trees in the LiDAR image are less distinct than the trees in the orthophotograph. The bottom left and right images are photographs taken on the ground at the center of the plot facing north and south respectively.

It is also possible to create three dimensional plots, in which the X and Y axes correspond to the X and Y coordinates of the pixels in the images, and the Z axis corresponds to either the DN values of the pixels or a vegetation index of a photographic image or the elevation values of a LiDAR canopy height model. Figure 1 shows these plots for an orthophotograph and a LiDAR CHM. From these plots it is evident that the tops of trees are concave, whereas the shadows surrounding the trees are convex. If a scene were examined in which the trees were surrounded by sparsely- or non-vegetated soil, a similar plot to the one constructed using the photographic DN values could be constructed by having the Z axis values correspond to the values of a vegetation index.

3.2.2 An algorithm that uses these generic attributes to locate trees and find their boundaries

The algorithm designed for this thesis uses these steps (illustrated in Figures 3, 4, 5, and 6):

1. When processing an aerial photograph, create a vegetation index image (Figure 5b), which is an image where higher DN values correspond to denser and / or healthier vegetation in the area covered by the pixel. The image is created by performing band arithmetic. Please refer to Section 3.2.3 and Procedure 1 for a list of vegetation indices that have been implemented, and details on the implementation. For a LiDAR canopy height model, the height of each pixel is used instead (Figure 6a).
2. Smooth the vegetation index image or height image using a 3x3 lowpass filter (Figures 5c and 6b; Procedure 2). This operation is performed to remove noise from the image.
3. Create local minima and local maxima images. These are produced using Laplacian (second derivative) of the Gaussian (LoG) filters (see Figure 7 and Procedure 3). The LoG is expressed as (continues on page 50):

$$G''(x) = \frac{(-0.5)2^{0.5} e^{-0.5x^2\sigma^{-2}}}{\pi^{0.5} \sigma^3} + \frac{0.5x^2 2^{0.5} e^{-0.5x^2\sigma^{-2}}}{\pi^{0.5} \sigma^5} \quad (1)$$

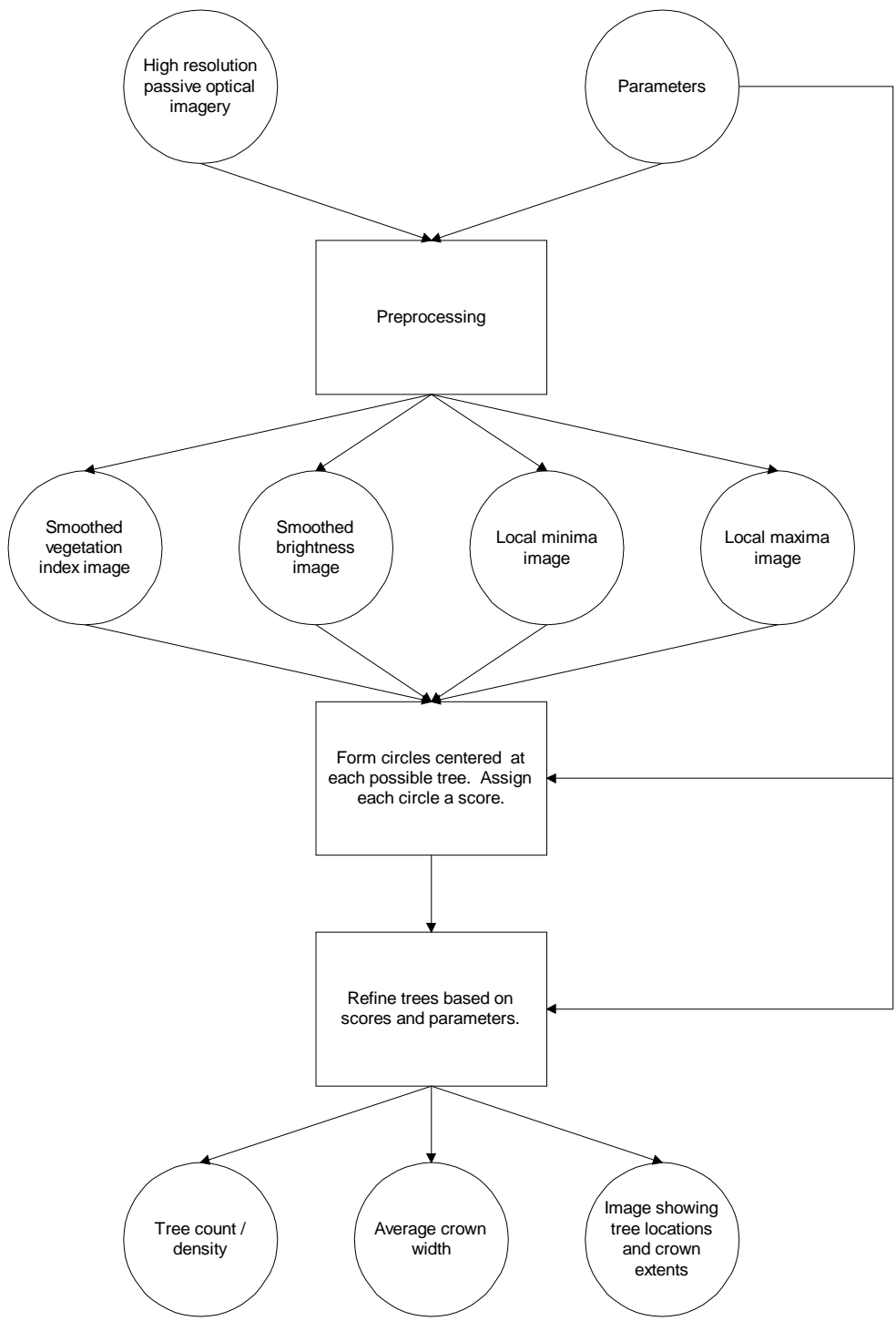


Figure 3. A flow chart showing the basic steps used by the computer vision algorithm described in Section 3.2.2 for processing a digital aerial photograph or other high resolution passive optical image.

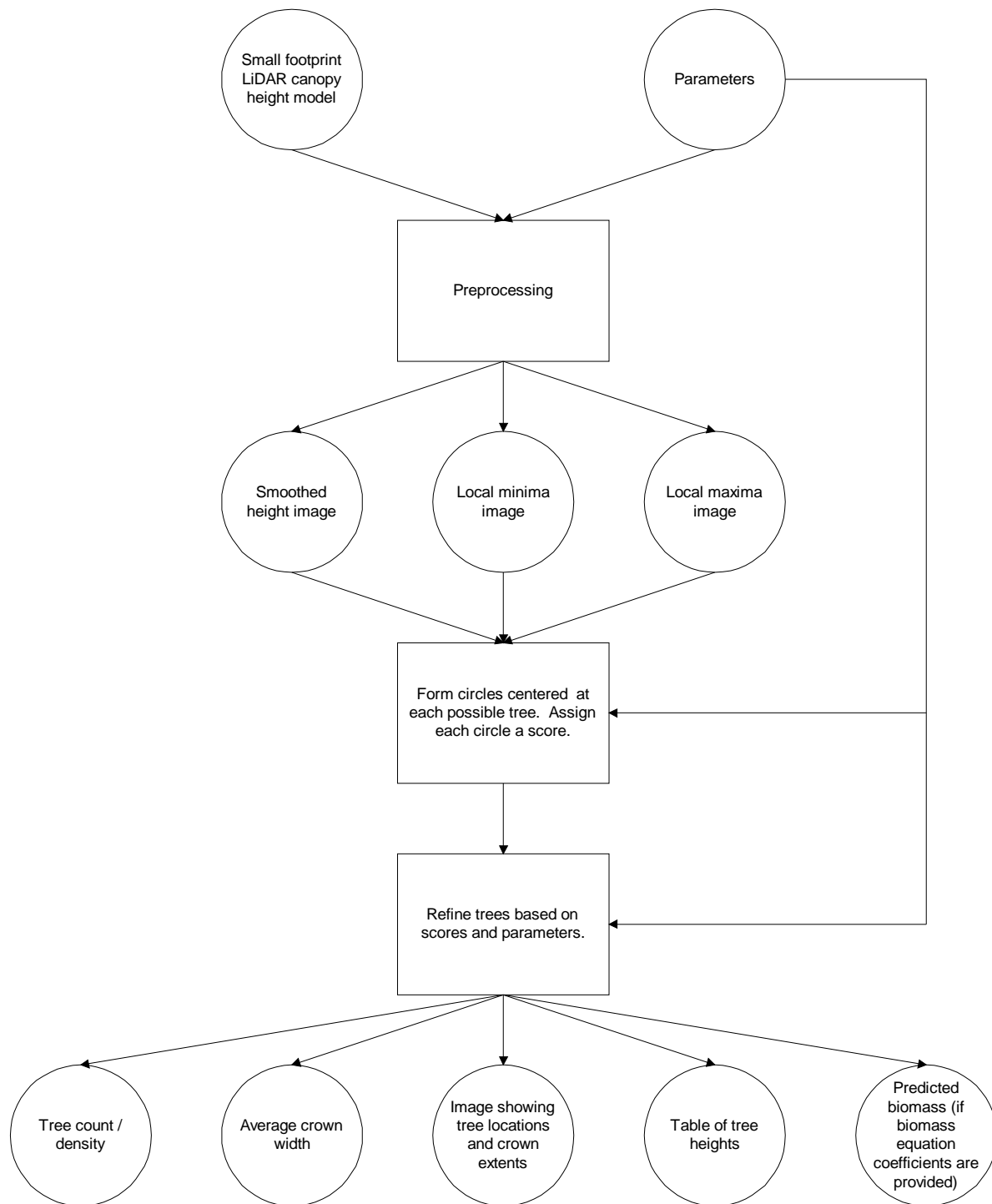


Figure 4. A flow chart showing the basic steps used by the computer vision algorithm described in Section 3.2.2 for processing a canopy height model (CHM) derived from small footprint LiDAR imagery.

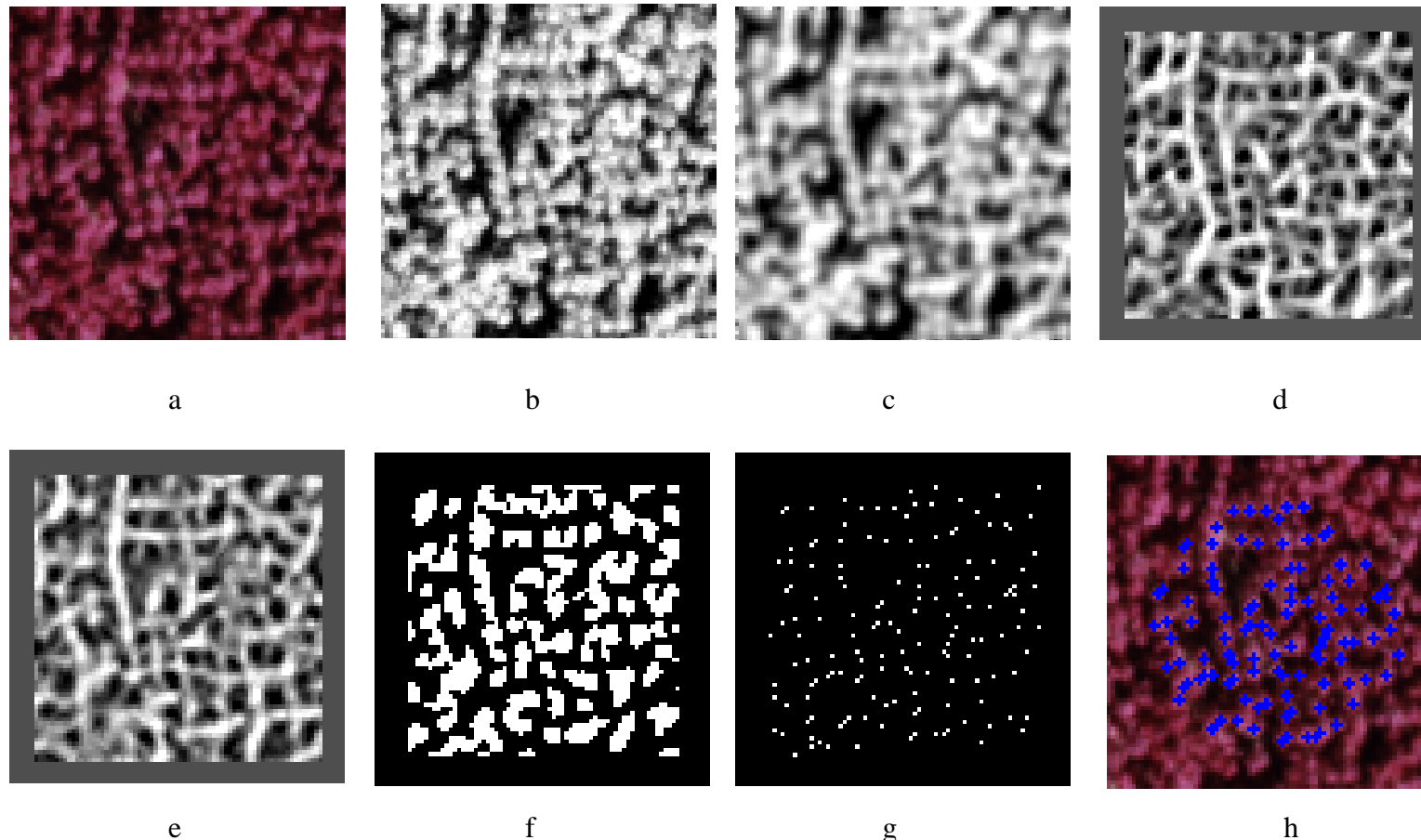


Figure 5. Steps of the tree finding algorithm (aerial photograph). These images show the intermediate images produced while processing a single image subset (in this case, the subset of plot 1: See Section 3.4). The images are: a) the original image; b) the vegetation index image; c) the vegetation index image after smoothing with a low pass filter; d) the local minimum image; e) the local maximum image; f) the image showing the locations of possible tree tops; g) the image showing the pixels most likely to represent the tops of actual trees; h) the locations of trees in the plot found by the program (blue crosses) overlain on the original image. Images d - e contain unprocessed border areas corresponding to areas in which the full width of the local minimum filter could not be applied.

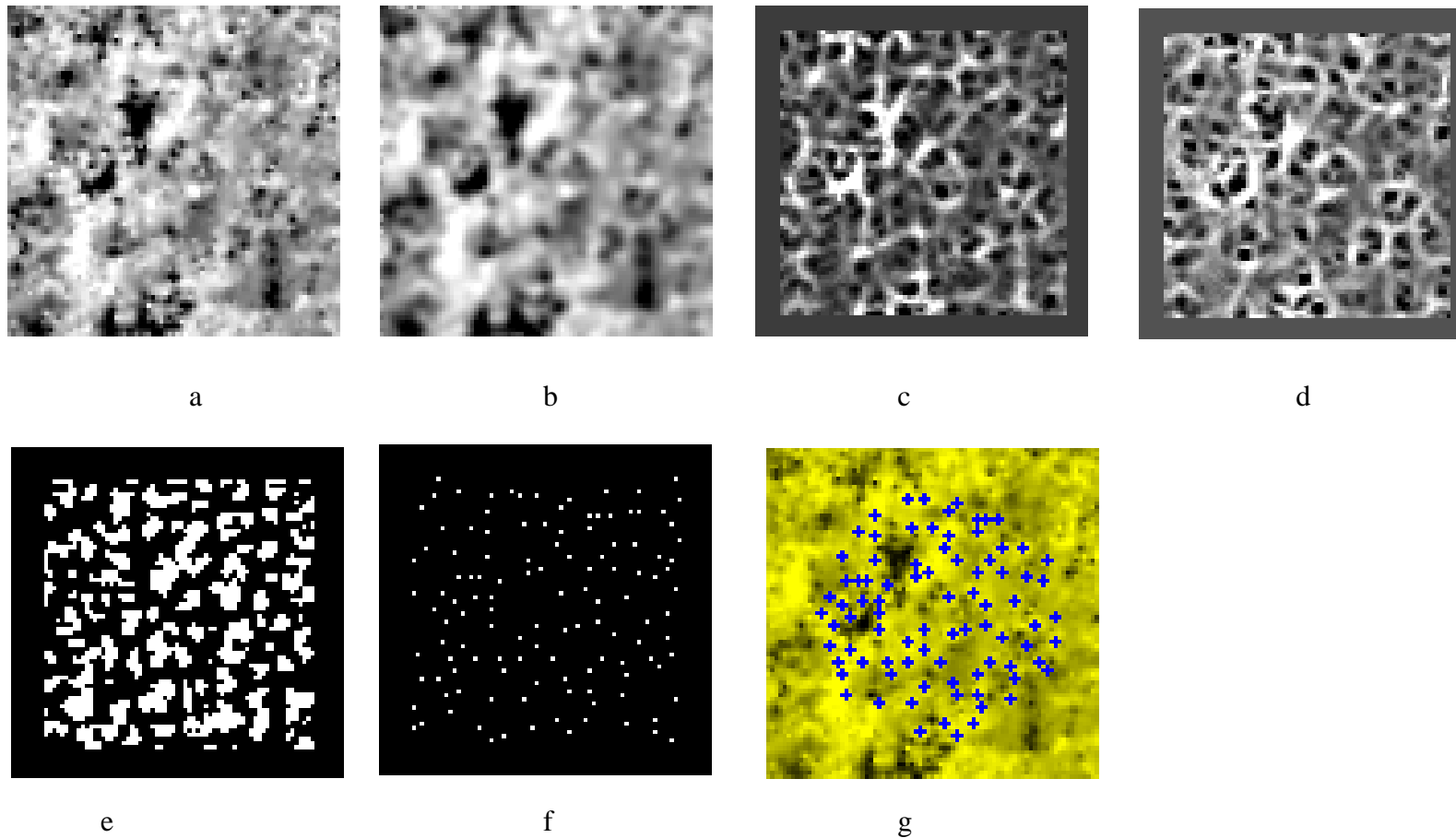


Figure 6. Steps of the tree finding algorithm (LiDAR CHM). These images show the intermediary images produced while processing a single image subset (in this case, the subset of plot 1: see Section 3.4). The images are: a) the original image; b) the original image after being smoothed by a low pass filter; c) the local minimum image; d) the local maximum image; e) the image showing the locations of possible tree tops; f) the image showing the pixels most likely to represent the tops of actual trees; g) the locations of trees in the plot found by the program (blue crosses) overlain on the original image. Images c - f contain unprocessed border areas corresponding to areas in which the full width of the local minimum filter could not be applied.

Procedure 1. Pseudocode for the vegetation index creation procedure. In this procedure, mathematical operations are performed on a three band image, *image*, to produce a new vegetation index image, *ratio_image*. The type of vegetation index is specified by the *ratio_method* parameter. Please see section 3.2.3 for a description of the vegetation indices corresponding to the parameter values.

```
for lcv_row in range (0, image_ y_size):
    for lcv_column in range (0, image_x_size):
        value_1, value_2, value_3 = image.getpixel((lcv_column, lcv_row))
        if ratio_method == "D":
            ratio_value = value_2 - value_1
        if ratio_method == "C":
            ratio_value = value_1 - value_2
        if ratio_method == "N":
            if value_1 + value_2 > 0:
                ratio_value = (value_1 - value_2) / (value_1 + value_2)
            else:
                ratio_value = 0
        if ratio_method == "I":
            ratio_value = value_1
        if ratio_method == "G":
            ratio_value = value_2
        ratio_image.putpixel((lcv_column, lcv_row), ratio_value)
return ratio_image
```

Procedure 2. Pseudocode for a procedure that performs a 3x3 low pass filter on a single band image, *image*, to produce a filtered image, *lp_image*.

```
for lcv_row in range (0, y_image_size):
    for lcv_column in range (0, x_image_size):
        if lcv_row == 0 or lcv_row == y_image_size - 1 or lcv_column == 0 or
           lcv_column == x_image_size:
            value =image.getpixel(lcv_column, lcv_row)
            lp_image.putpixel((lcv_column, lcv_row), value)
        else:
            value = image.getpixel(lcv_column, lcv_row)
            value = value + image.getpixel(lcv_column - 1, lcv_row - 1)
            value = value + image.getpixel(lcv_coumns, lcv_row - 1)
            value = value + image.getpixel(lcv_column + 1, lcv_row - 1)
            value = value + image.getpixel(lcv_column - 1, lcv_row)
            value = value + image.getpixel(lcv_column + 1, lcv_row)
            value = value + image.getpixel(lcv_column - 1, lcv_row + 1)
            value = value + image.getpixel(lcv_column, lcv_row + 1)
            value = value + image.getpixel(lcv_column+1, lcv_row + 1)
            value = value / 9
            lp_image.putpixel((lcv_column, lcv_row), value)
return lp_image
```

Procedure 3. Pseudocode for using LoG filters to obtain local minimum (*lmin_image*) and local maximum (*lmax_image*) images based on a single band input image, *image*. The standard deviation used for the LoG filter is contained in the *sd* parameter.

```

multipliers = []
for lcv_x in range(-5, 6):
    term_1 = (-0.5 * 2^0.5) / (pi^0.5 * sd^3)
    term_2 = exp(-0.5 * ((lcv_x ^2) / (sd^2)))
    term_3 = (0.5 * 2^0.5) / (pi^0.5 * sd^5)
    term_4 = lcv_x^2
    term_5 = exp(-0.5 * (lcv_x ^2) / (sd^2))
    coef = (term_1 * term_2) + (term_3 * term_4 * term_5)
    multipliers.append(coef)

for lcv_row in range(6, y_image_size - 6):
    for lcv_column in range(6, x_image_size - 6):
        lm_1 = 0
        lm_2 = 0
        lm_3 = 0
        lm_4 = 0
        for lcv_local in range(-6, 7):
            lm_1 = lm_1 _ image.getpixel(lcv_column, lcv_row + lcv_local) * coef[abs(lcv_local)]
            lm_1 = lm_1 + image.getpixel(lcv_column-1, lcv_row + lcv_local) * coef[abs(lcv_local)]
            lm_1 = lm_1 + image.getpixel(lcv_column+1, lcv_row + lcv_local) * coef[abs(lcv_local)]
            lm_2 = lm_2 + image.getpixel(lcv_column + lcv_local, lcv_row) * coef[abs(lcv_local)]
            lm_2 = lm_2 + image.getpixel(lcv_column + lcv_local, lcv_row + 1) * coef[abs(lcv_local)]
            lm_2 = lm_2 + image.getpixel(lcv_column + lcv_local, lcv_row - 1) * coef[abs(lcv_local)]
            lm_3 = lm_3 + image.getpixel(lcv_column + lcv_local, lcv_row + lcv_local) * coef[abs(lcv_local)]
            lm_3 = lm_3 + image.getpixel(lcv_column + lcv_local - 1, lcv_row + lcv_local + 1) * coef[abs(lcv_local)]
            lm_3 = lm_3 + image.getpixel(lcv_column + lcv_local + 1, lcv_row + lcv_local - 1) * coef[abs(lcv_local)]
            lm_4 = lm_4 + image.getpixel(lcv_column - lcv_local, lcv_row + lcv_local) * coef[abs(lcv_local)]

```

```
lm_4 = lm_4 + image.getpixel(lcv_column - lcv_local - 1, lcv_row + lcv_local - 1) * coef[abs(lcv_local)]
lm_4 = lm_4 + image.getpixel(lcv_column - lcv_local + 1, lcv_row + lcv_local + 1) * coef[abs(lcv_local)]
lmax = minimum(lm_1, lm_2, lm_3, lm_4)
lmax_image.putpixel((lcv_column, lcv_row), lmax)
lmin = maximum(lm_1, lm_2, lm_3, lm_4)
lmin_image.putpixel((lcv_column, lcv_row), lmin)
```

Laplacian of the Gaussian (Variance = 1.5)

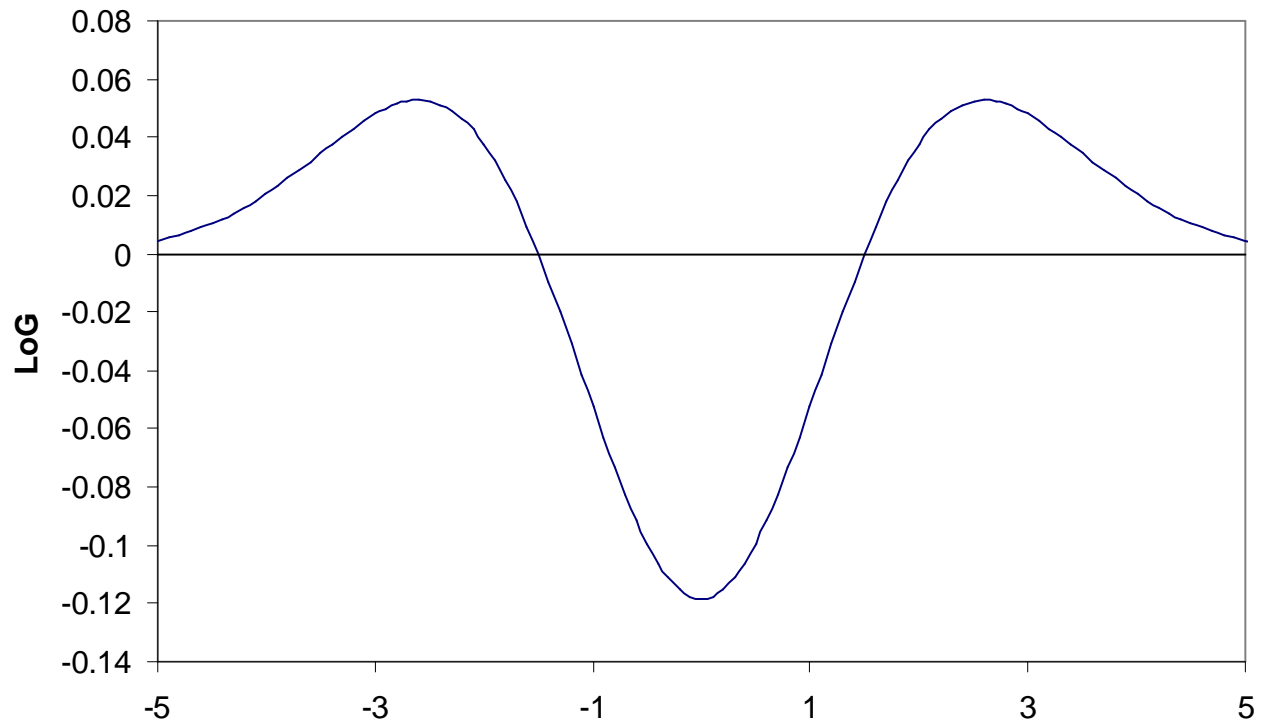


Figure 7. The Laplacian of the Gaussian used for finding local minima.

Where:

$G''(x)$ is the LoG of x

σ is the standard deviation

LoG filters have a strong basis in biological vision, since work has shown the receptive fields in eyes have an excitatory central portion surrounded by an inhibitory portion (Fiorentini *et al.*, 1990), like the LoG. Additionally, several contemporary vision theories indicate that the response of the eye to boundaries and / or line objects has a shape similar to the LoG (Fiorentini *et al.*, 1990). The LoG filters used in this work are one dimensional, and oriented in the X and Y directions plus the two main diagonals. In order to reduce noise, the filters have a width of three pixels. The variance used in computing the LoG is variable. To find the value of the local minima, the four LoG filters are passed

over either the non-smoothed vegetation index image or a single band from the original image (aerial photograph) or the non-smoothed LiDAR height values. The maximum filtered value for each pixel is then taken, and assumed to be the local minimum (Figures 5d and 6c), and the negative of the minimum filtered value is considered to be the local maximum (Figures 5e and 6d).

4. For each vegetated pixel in the image (based on the vegetation index or height image; see Figures 5f and 6e) that is not a local minimum, circles with different radii are formed. These circles are centered at the pixel, and are formed by expanding the radius outward one pixel at a time. The range of radii is based upon the minimum and maximum crown diameters that the user expects to find in the image. Each circle is assigned a score, which corresponds to its likelihood of representing the boundary of a tree. This score depends on the average of the local minima image underlying the circle's boundary and the radius (larger circles can be penalized or favored.). The radius with the highest score is chosen as the most probable radius for the pixel being examined. Additionally, expansion of the circle can be halted if the circle encounters an exceptionally dark / low pixel, or if a pixel that is unlikely to be vegetated (corresponding to bare or sparsely vegetated soil) is encountered (aerial photographs only). Procedures 4 and 5 provide pseudocode for this procedure.

Once the circles associated with each pixel have been selected, a refinement procedure is used to choose which pixels actually represent the tops of trees. The principle used is that the center of one tree cannot be located within the crown of another tree. To implement this, a score is assigned to each vegetated pixel. The score is based on 1) how concave the area around the center pixel is (whether it is a local maxima), 2) how convex the area around the pixels underlying the circle associated with the center pixel are and 3) how bright or high the center pixel is. After the scores have been computed, the algorithm examines each vegetated pixel to determine whether it should be considered to be the top of a tree. To determine whether it is kept, the score of the pixel being examined is compared to the scores of pixels underlying the circle associated with the examined pixel. If any of these pixels have a higher value, the pixel being examined is considered to not represent the top of a tree. This refinement process is run twice in order to

eliminate any small trees missed during the first pass. After completion, the remaining pixels should represent the tops of actual trees, and the circles associated with these pixels should correspond to their crown boundaries (Figures 5g, 5h, 6f, and 6g). For the LiDAR imagery, the height of each tree is considered to be the greatest height within each circle. Please refer to Procedures 5 and 6 for details of this procedure.

For some applications, it is desirable to obtain biomass estimates as well as the number and locations of trees. Two methods of doing this were evaluated.

The first method involves obtaining the tree density for a user defined area from either the aerial photographs or the LiDAR canopy height model. Next, the heights of the trees obtained from the LiDAR image are ranked from highest to lowest, and the heights of the tree at the 25, 50 and 75th percentile are extracted. Two equations are then formed using stepwise multiple linear regression ($p \leq 0.05$ to enter, $p \leq 0.1$ to remain). In the first equation, stand biomass is the dependent variable, and the independent variables considered are the ones listed in the first column of Table 4. In the second equation, the natural log of the stand biomass is the dependent variable, and the independent variables considered are the ones listed in the second column of Table 4. The best equation was selected by comparing the untransformed r^2 values for the two equations.

This approach follows the basic approach Lefsky *et al.* (1999a) and Drake *et al.* (2002) in which a large number of metrics are derived, and stepwise multiple linear regression is used to create a model to predict the biophysical parameter of interest. The metrics used in this study were chosen because both height and density (based on the number of trees per plot) are related to biomass, and the height of the canopy at the 25, 50, and 75th percentiles should reflect structural differences among stands that equate to biomass differences (continues on page 62).

Procedure 4. Pseudocode for the orthophotograph-based form of a function that locates possible trees in an image and attempts to find their diameters. To do this, the local minimum image, *lmin_image*, a vegetation index image, *ratio_image*, and smoothed version of the image used for producing the local minimum image, *edge_image*, are used. The product of this function are two images: *value_image* which contains the sum of the pixels in *lmin_image* underlying each possible tree, and *radius_image*, which contains the crown radius associated with each possible tree. Please Section 3.2.2 for a high level description on this procedure, and Section 3.2.3 for a description of the parameters that are used.

```
if min_total < 0 then min_total = 0
```

```
for lcv_line in range (max_radius, y_image_size - max_radius):
```

```
    for lcv_column in range (max_radius, x_image_size - max_radius):
```

```
        test_1 = lmin_image(lcv_column, lcv_line)
```

```
        test_2 = ratio_image(lcv_column, lcv_line)
```

```
        if test_1 < center_thresh and test_2 > ratio_thresh:
```

```
            high_value = -999
```

```
            high_radius = 0
```

```
            stop = "N"
```

```
            stop_now = "N"
```

```
            for lcv_radius in range(min_radius, max_radius + 1):
```

```
                if stop == "Y":
```

```
                    stop_now = "Y"
```

```
                pixels = 0
```

```
                value = 0
```

```
                for lcv_y in range (-1 * lcv_radius, lcv_radius + 1):
```

```
                    x_radius = (lcv_radius^2 - lcv_y ^2)^0.5
```

```
                    position_y = lcv_line + lcv_y
```

```
                    pos_position_x = lcv_column + x_radius
```

```
                    neg_position_x = lcv_column - x_radius
```

```
                    if ratio_image.getpixel(pos_position_x, position_y) < adj_thresh or
```

```
                        edge_image.getpixel(pos_position_x, position_y) < ae_thresh:
```

```

        stop = "Y"
    if ratio_image.getpixel(neg_position_x, position_y) < adj_thresh or
        edge_image.getpixel(neg_position_x, position_y) < ae_thresh:
        stop = "Y"
    value = value + lmin_image.getpixel(pos_position_x, position_y)
    value = value + lmin_image.getpixel(neg_position_x, position_y)
    if x_radius == 0:
        pixels = pixels + 1
    else:
        pixels = pixels + 2

total_value = value / pixels
total_value = total_value + decay*lcx_radius
if total_value > high_value:
    high_value = total_value
    high_radius = lcx_radius
if high_value < min_total:
    high_value = 0
if stop_now == "Y":
    break

value_image.putpixel((lcx_column, lcx_line), high_value)
radius_image.putpixel((lcx_column, lcx_line), high_radius)

```

Procedure 5. Pseudocode for the LiDAR-based form of a function that locates possible trees in an image and attempts to find their diameters. To do this, the local minimum image, *lmin_image*, the unsmoothed CHM, *raw_image*, and a smoothed CHM, *smoothed_image* are used. The product of this function are two images: *value_image* which contains the sum of the pixels in *lmin_image* underlying each possible tree, and *radius_image*, which contains the crown radius associated with each possible tree. Please Section 3.2.2 for a high level description on this procedure, and Section 3.2.3 for a description of the parameters that are used.

```

if min_total < 0:
    min_total = 0

for lcv_line in range (max_radius, y_image_size - max_radius):
    for lcv_column in range (max_radius, x_image_size - max_radius):
        test_1 = lmin_image.getpixel(lcv_column, lcv_line)
        test_2 = raw_image.getpixel(lcv_column, lcv_line)
        if test_1 < center_thresh and test_2 > ratio_thresh:
            high_value = -999
            high_radius = 0
            stop = "N"
            stop_now = "N"
            for lcv_radius in range(min_radius, max_radius + 1):
                if stop == "Y":
                    stop_now = "Y"
                pixels = 0
                value = 0
                for lcv_y in range (-1 * lcv_radius, lcv_radius + 1):
                    x_radius = (lcv_radius^2 - lcv_y ^2)^0.5
                    position_y = lcv_line + lcv_y
                    pos_position_x = lcv_column + x_radius
                    neg_position_x = lcv_column - x_radius
                    if (test_2 -lp_image.getpixel(pos_position_y, position_y)) > ae_thresh:

```

```

        stop = "Y"
    if (test_2 - lp_image.getpixel(neg_position_x, position_y)) > ae_thresh:
        stop = "Y"
    value = value + lmin_image.getpixel(pos_position_x, position_y)
    value = value + lmin_image.getpixel(neg_position_x, position_y)
    if x_radius == 0:
        pixels = pixels + 1
    else:
        pixels = pixels + 2

total_value = value / pixels
total_value = total_value + decay*lcv_radius
if total_value > high_value:
    high_value = total_value
    high_radius = lcv_radius
if high_value < min_total:
    high_value = 0
if stop_now == "Y":
    break

value_image.putpixel((lcv_column, lcv_line), high_value)
radius_image.putpixel((lcv_column, lcv_line), high_radius)

```

Procedure 6. Pseudocode for the orthophoto-based form of the procedure of eliminating trees found by Procedure 4 that are unlikely to correspond to actual trees. To accomplish this, the output images from Procedure 4 are used (*radius_image* and *value_image*), together with the local maximum image (*lmax_image*), and the vegetation index image (*ratio_image*).

```

for lcv_line in range (max_radius, y_image_size - max_radius):
    for lcv_column in range (max_radius, x_image_size - max_radius):
        pixel_rad = radius_image.getpixel(lcv_column, lcv_line)
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_image.getpixel(lcv_column, lcv_line)
            score = score + beta * lmax_image.getpixel(lcv_column, lcv_line)
            score = score + gamma * ratio_image.getpixel(lcv_column, lcv_line)
            for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                    radius = ((win_y - lcv_line)^2 + (win_x - lcv_column)^2)^0.5
                    ignore = "N"
                    if radius > pixel_rad:
                        ignore = "Y"
                    if ignore == "N":
                        wscore = alpha * value_image.getpixel(win_x, win_y)
                        wscore = wscore + beta * lmax_image.getpixel(win_x, win_y)
                        wscore = wscore + gamma * ratio_image.getpixel(win_x, win_y)
                        if wscore > score:
                            keep = 0
                            break
                if int(keep) == 0:
                    break
            images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

for lcv_line in range (max_radius, y_image_size - max_radius):

```

```

for lcv_column in range (max_radius, x_image_size - max_radius):
    pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
    if pixel_rad > 0:
        keep = 1
        score = alpha * value_image.getpixel(lcv_column, lcv_line)
        score = score + beta * lmax_image.getpixel(lcv_column, lcv_line)
        score = score + gamma * ratio_image.getpixel(lcv_column, lcv_line)
        for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
            for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                radius = ((win_y - lcv_line)^2 + (win_x - lcv_column)^2)^0.5
                ignore = "N"
                if radius > pixel_rad:
                    ignore = "Y"
                if ignore == "N" and radius_image.getpixel(win_x, win_y) > 0:
                    wscore = alpha * value_image.getpixel(win_x, win_y)
                    wscore = wscore + beta * lmax_image.getpixel(win_x + win_y)
                    wscore = wscore + gamma * ratio_image.getpixel(win_x, win_y)
                    if wscore < score:
                        images.radius_image.putpixel((win_x, win_y), 0)

```

Procedure 7. Pseudocode for the LiDAR form of the procedure of eliminating trees found by Procedure 5 that are unlikely to correspond to actual trees. To accomplish this, the output images from Procedure 5 are used (*radius_image* and *value_image*), together with the local maximum image (*lmax_image*), and the vegetation index image (*ratio_image*).

```

for lcv_line in range (max_radius, y_image_size - max_radius):
    for lcv_column in range (max_radius, x_image_size - max_radius):
        pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_image.getpixel(lcv_column, lcv_line)
            score = score + beta * lmax_image.getpixel(lcv_column, lcv_line)
            score = score + gamma * raw_image(lcv_column, lcv_line)
            for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                    radius = ((win_y - lcv_line)^2 + (win_x - lcv_column)^2)^0.5
                    ignore = "N"
                    if round(radius) > pixel_rad:
                        ignore = "Y"
                    if ignore == "N":
                        wscore = alpha * value_image.getpixel(win_x, win_y)
                        wscore = wscore + beta * lmax_image.getpixel(win_x, win_y)
                        wscore = wscore + gamma * raw_image.getpixel(win_x, win_y)
                        if wscore > score:
                            keep = 0
                            break
                if keep == 0:
                    break
            radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

for lcv_line in range (max_radius, y_image_size - max_radius):

```



```

for lcv_column in range (max_radius, x_image_size - max_radius):
    pixel_rad = radius_image.getpixel(lcv_column, lcv_line)
    if pixel_rad > 0:
        keep = 1
        score = alpha * value_image.getpixel(lcv_column, lcv_line)
        score = score + beta * lmax_image.getpixel(lcv_column, lcv_line)
        score = score + gamma * raw_image.getpixel(lcv_column, lcv_line)
        for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
            for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                radius = ((win_y - lcv_line)^2 + (win_x - lcv_column)^2) ^ 0.5
                ignore = "N"
                if round(radius) > pixel_rad:
                    ignore = "Y"
                if ignore == "N" and radius_image.getpixel(win_x, win_y) > 0:
                    wscore = alpha * value_image.getpixel(win_x, win_y)
                    wscore = wscore + beta * lmax_image.getpixel(win_x, win_y)
                    wscore = wscore + gamma * raw_image.getpixel(win_x, win_y)
                    if wscore < score:
                        images.radius_image.putpixel((win_x, win_y), 0)

```

Table 4. Independent variables considered in the stepwise multiple linear regression used to create equations for the first biomass prediction technique described in Section 3.2.2. In the table, B is the stand biomass, T is the number of trees obtained by the computer vision algorithm, and H25, H50, and H75 are the heights of the trees in each plot at the 25th, 50th, and 75th percentiles.

Dependent variable: Stand biomass	Dependent variable: ln(Stand biomass)
T	ln(T)
T ²	[ln(T)] ²
T ^{0.5}	[ln(T)] ^{0.5}
H25	ln(H25)
H50	ln(H50)
H75	ln(H75)
H50 / H75	ln(H50 / H75)
H75 / H50	ln(H75 / H50)
H75 / H25	ln(H75 / H25)
T • H25	ln(T • H25)
T • H50	ln(T • H50)
T • H75	ln(T • H75)
T • (H50) ²	[ln(T • H50)] ²
(H25) ²	[ln(H25)] ²
(H50) ²	[ln(H50)] ²
(H75) ²	[ln(H75)] ²
(H25) ^{0.5}	[ln(H25)] ^{0.5}
(H50) ^{0.5}	[ln(H50)] ^{0.5}
(H75) ^{0.5}	[ln(H75)] ^{0.5}

The second method is based on determining the biomasses of individual trees and summing the biomasses to obtain a stand-level estimate. To do this, the regression form of a common biomass equation form known as the logarithmic form was used (Clutter *et al.*, 1983, p. 8). However, stand density was used in place of DBH because DBH cannot be measured directly from the imagery. Substituting stand density for DBH is reasonable because researchers have found that DBH generally increases as stand density decreases (e.g., Cain and Shelton, 2003). The resulting equation is:

$$\ln(B) = \alpha \ln(H) + \beta \ln(D) + \gamma \quad (2)$$

Where:

B is the biomass of the individual tree

H is the individual tree height

D is the density of the stand the trees are in

α , β , γ are coefficients that must be estimated

Unfortunately, it is uncommon to know the biomass values of specific trees in an image, so this equation cannot be used directly when estimating the coefficients for it. Instead, plot level biomass values must be used. This is done by summing Equation 2 for all trees in the training plots based on image-derived values (Equation 3). Coefficients (α , β , γ) are found using the Nelder-Mead simplex algorithm (see Section 3.2.4.1.1) together with the plot height and density values from the training images and the plot biomass values that were measured on the ground. It is necessary to use an optimization routine such as the Nelder-Mead simplex algorithm because the system of equations is non-linear and cannot be linearized.

$$\begin{aligned}
SB_1 &= \sum_{i=1}^{N_1} e^{[\alpha \ln(H_i) + \beta \ln(D_i) + \gamma]} \\
&\vdots \\
SB_n &= \sum_{i=1}^{N_{11}} e^{[\alpha \ln(H_i) + \beta \ln(D_i) + \gamma]}
\end{aligned} \tag{3}$$

Where:

SB_j is the above ground biomass of the j^{th} plot

N_j is the number of trees found in the j^{th} plot

n is the number of plots used for optimization

i is a counting variable

H and D are the same as in Equation 2

The starting point for the optimization was obtained by fitting Equation 2 to individual tree data obtained for twelve year old loblolly pines in the Appomattox-Buckingham State Forest as part of a spacing trial conducted by the Loblolly Pine Growth and Yield Research Cooperative at the Virginia Polytechnic Institute and State University.

3.2.3 Parameter descriptions and initial values

Parameters that are optimized during training have been noted with an asterisk. For these parameters, the minimum and maximum values specified at the start of training are listed in brackets. For non-optimized parameters, the parameter value selected has been noted in brackets after the parameter name. The parameter names are identical to the variable and constant names used in Procedures 1 - 7, and in Appendices A - F.

Ratio method [I]: This parameter specifies which method should be used to create the image for to determine whether a pixel is vegetated or not. This parameter applies only to orthophotos.

1.D:

A simple vegetation index designed for normal color images. It is calculated by subtracting the DN values in the red band from the DN values in the green band.

2. C:

A vegetation index designed for false color images. The DN value in the red band is subtracted from the DN value in the near infrared band.

3. N:

The normalized difference vegetation index (NDVI), which is a common vegetation index used with false color imagery. It is calculated using the equation:

$$NDVI = \frac{DN_{NIR} - DN_R}{DN_{NIR} + DN_R} \quad (4)$$

Where

DN_{NIR} is the DN value in the near infrared and DN_R is the DN value in the red.

4. I:

Uses the near infrared DN values from a color infrared photograph.

5. G:

Uses the green DN values from a normal color image.

Edge image [IR (CIR)]:

The image used when finding the local minima in the image. This parameter applies only to orthophotos.

1. Ratio

Uses the ratio image to find the local minima. This option is useful when trees are surrounded by bare soil or another cover other than green vegetation.

2. IR (CIR)

Uses the DN values in the IR band of a color infrared image. This option is useful if the trees form a closed canopy, since it is sensitive to the presence of shadows.

3. GREEN (NC)

Uses the green DN values from a normal color image. This is a good option in images in which the trees form a closed canopy, since it is sensitive to the presence of shadows.

*SD [1, 2.5]:

The standard deviation used to compute the LoG.

Min_radius [1]

The minimum expected tree radius in the image, in pixels.

Max_radius [8]

The maximum expected tree radius, in pixels.

*Ratio_thresh [-5, 20] (air photo), [2, 12] (LiDAR)

This parameter is used to prevent the program from searching for trees in areas that do not contain green vegetation. If the value of the ratio image (orthophoto) or height of the CHM (LiDAR) is lower than the value specified, the program will not try to find a tree centered at the pixel.

*Center_thresh [-5, 10] (orthophoto), [0, 10] (LiDAR)

This parameter is used to prevent the program from finding trees centered at local minima. If the value of the local minima image is greater than the value specified, the program will not try to find a tree centered at this pixel.

*Min_total [0, 10]

During the refinement stage, one of the criteria used to determine a potential tree top's score is how concave up the pixels underlying the tree top's boundary are. Through experimentation, it was found that the algorithm performed better if potential trees were rewarded for having crowns underlain by pixels that were highly concave up, but that having crowns underlain by pixels that are only slightly concave up or are concave down should not count against the score. This is accomplished by setting this threshold, which determines how concave up the pixels under the perimeter must be before they increase the score. The reason this parameter is beneficial may be that trees that closely border neighboring trees are not separated by deep shadows or consistently low areas. As a result these trees may be improperly eliminated if potential tree tops' scores are excessively penalized for not having boundaries underlain by concave up pixels.

*Adj_thresh [-10, 20]

Orthophoto only. If any of the pixels in the radius being examined have a ratio value that is less than the value specified by this parameter, a flag is set. The flag causes the program to stop expanding the radius of the circle after one more iteration.

*Ae_thresh [20, 90] (air photo), [0, 10] (LiDAR)

For orthophotos, this parameter is similar to the adjacent ratio threshold. The difference is that the DN values in the image used for finding the local minima rather than the ratio image is examined. The value of this parameter is dependent on the edge image you use and on the individual image. For LiDAR imagery, the difference between the center pixel and the perimeter pixel in the CHM is evaluated, and expansion is halted if this value is exceeded.

*Decay [-20, 10]

In some cases the program will tend to either clump trees together or break trees apart. This parameter addresses this by increasing or decreasing the average of the local minima

by an amount proportional to the radius in pixels. A positive value will cause an increase, and a negative value will cause a decrease.

*Alpha, beta and gamma parameters:

Parameters used in the refinement stage of the processing in order to assign scores to each pixel.

1. The alpha parameter [0, 5]:

The alpha parameter is the importance assigned to the strength of the local minimum associated with the pixel.

2. The beta parameter [-5, 0]:

The beta parameter is the importance of the concavity of the DN values in the vicinity of the pixel.

3. The gamma parameter [0, 5]:

The gamma parameter is the importance of the DN value of the ratio image (orthophotos) or CHM (LiDAR) at the location of the pixel under examination.

3.2.4 Training

In order to set the parameters used to control the tree finding algorithm described in Section 3.2.3 (variance, ratio_thresh, center_thresh, min_total, adj_thresh, ae_thresh, decay, alpha, beta, gamma), an optimization algorithm is used in conjunction with training data consisting of small representative images in which the number of trees (and optionally the biomass of the trees) is known. Overviews of the procedures used for setting parameters and processing new data are illustrated in Figures 8 - 10. For an optimization routine to function, there must be a means of assessing the quality of the parameters. This is done using either Equation 5 or Equation 6, both of which are based on the error sum of squares (SSE), which is standard means of calculating the total error associated with a series of predictions (Weiss and Hassett, 1987, p. 522).

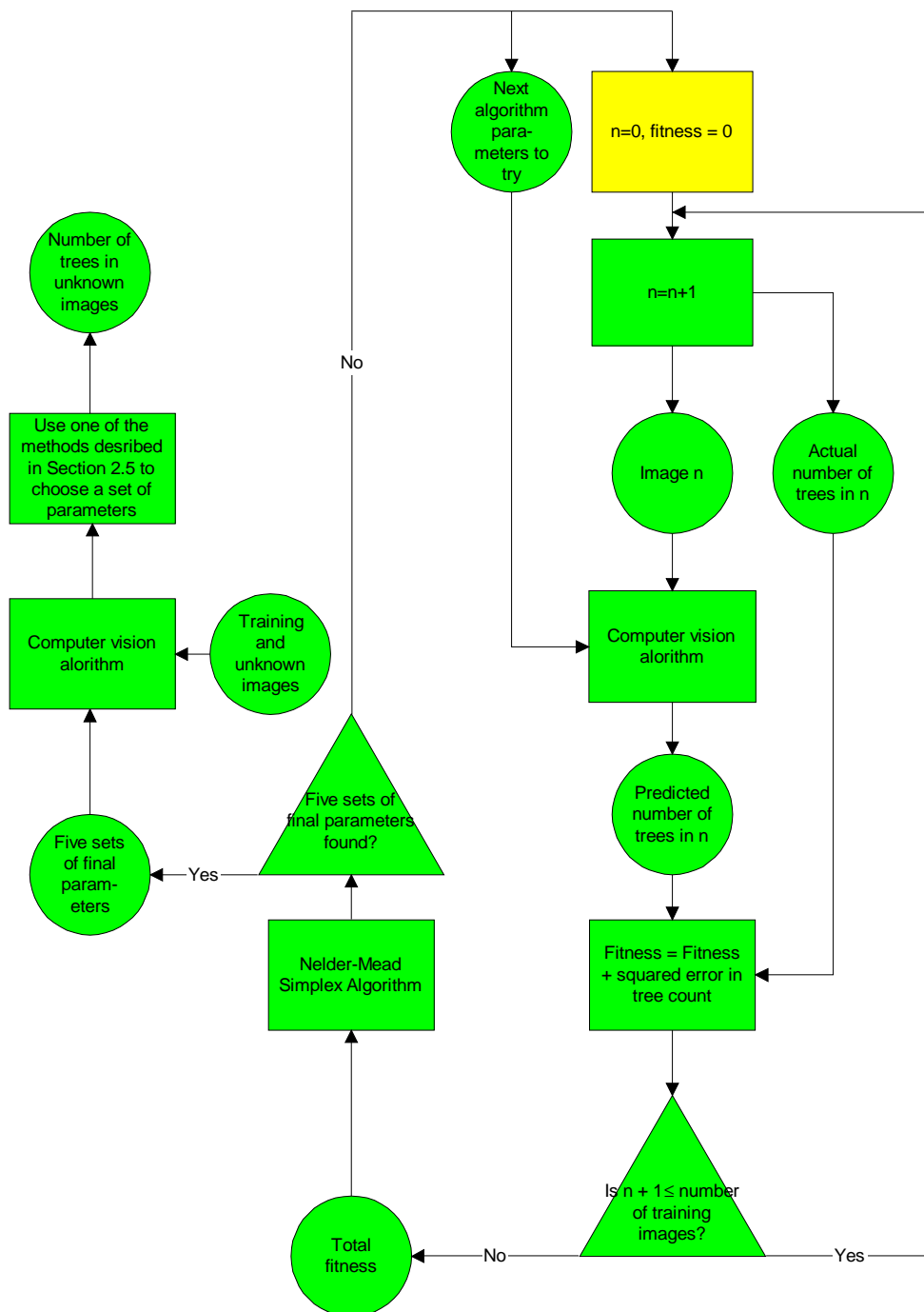


Figure 8. The processing flow used for processing orthophotographs or other high spatial resolution passive optical data, using training data to set the parameters for the computer vision algorithm. The starting point of the flow chart is indicated in yellow. Green steps indicate steps that are similar to (but not necessarily identical to) steps in Figures 9 and 10.

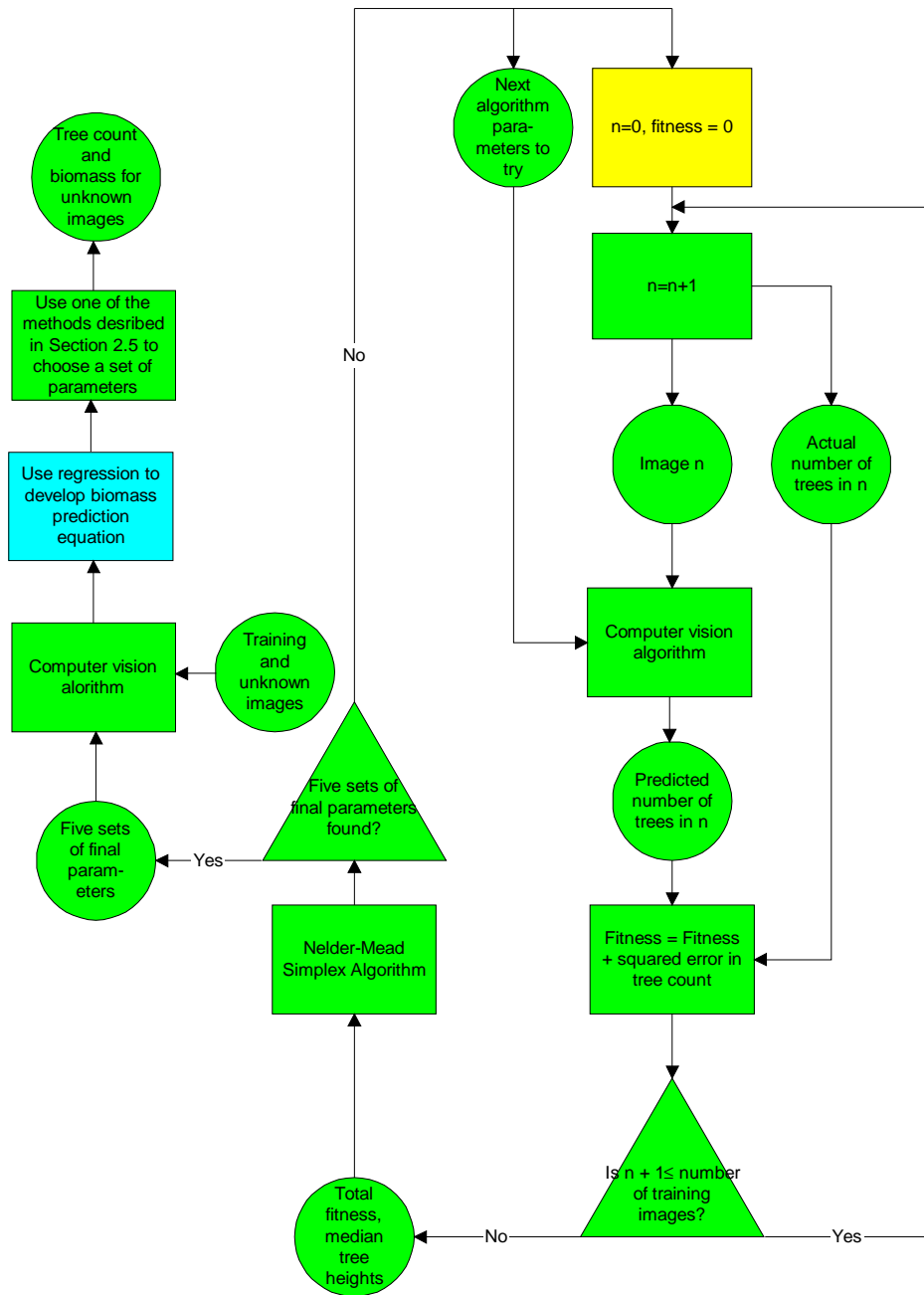


Figure 9. The processing flow used for processing LiDAR CHMs, using training data to set the parameters for the computer vision algorithm. In this case the biomass of each stand is determined using a regression equation that is derived after training is complete (the first biomass determination procedure described in Section 3.2.2). The starting point of the flow chart is indicated in yellow. Green steps indicate steps that are similar to (but not necessarily identical to) steps in Figures 8 and 10, whereas the cyan step is a step that is not similar to steps in Figures 8 and 10.

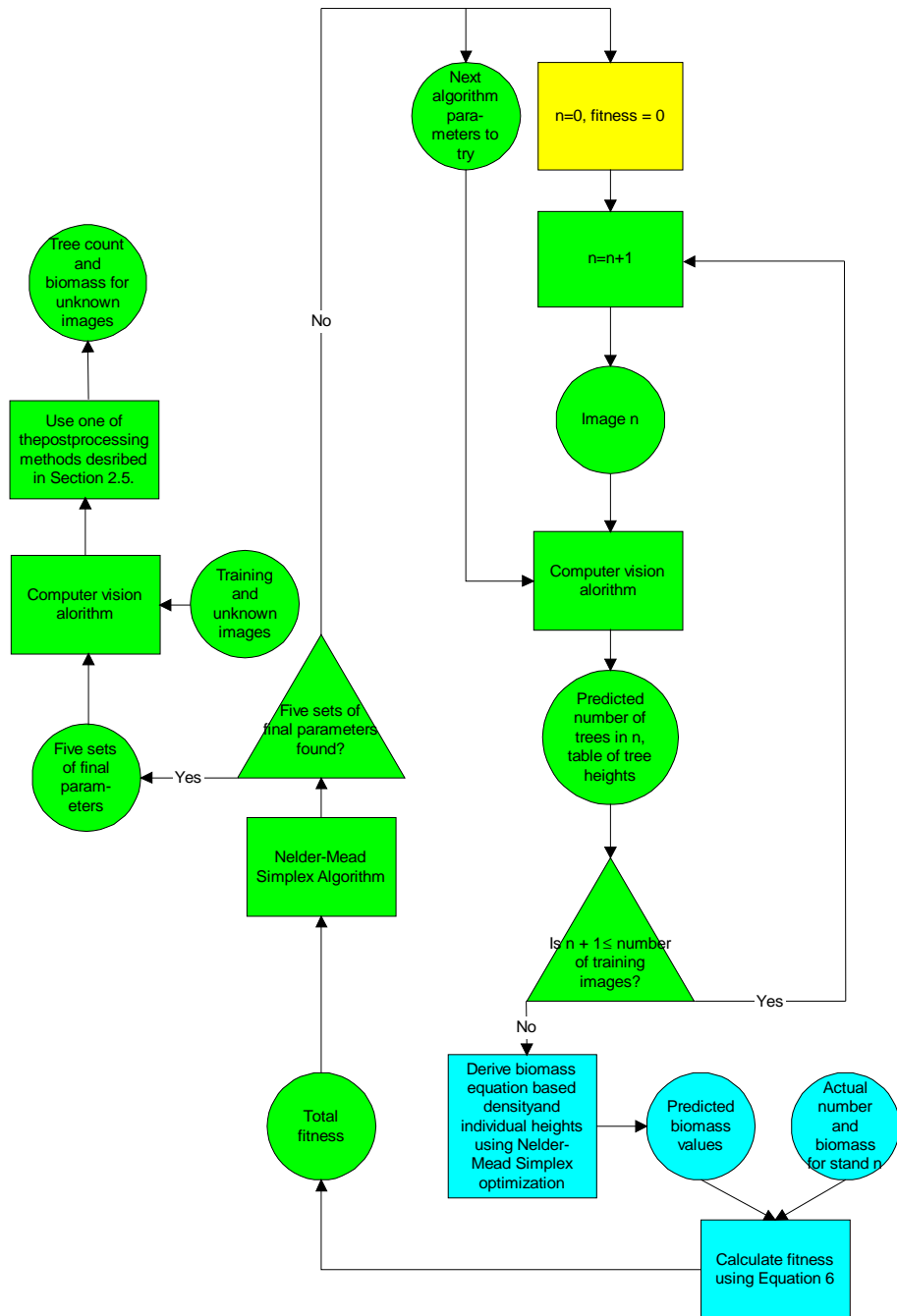


Figure 10. The processing flow used for processing LiDAR CHMs, using training data to set the parameters for the computer vision algorithm. In this case the biomass of individual trees is determined and summed (the second biomass determination procedure described in Section 3.2.2). The starting point of the flow chart is indicated in yellow. Green steps indicate steps that are similar to (but not necessarily identical to) steps in Figures 8 and 9, whereas the cyan steps are steps that are not similar to step in Figures 8 and 9.

$$F_i = \sum_{n=1}^{n=N} (T_{na} - T_{np})^2 \quad (5)$$

Where:

F_i is the fitness of the i^{th} individual solution being examined. The lower this value is, the more optimal the parameters are.

$$F_i = \sum_{n=1}^{n=N} \left((T_{na} - T_{np})^2 + (\text{relative_importance}) [S(B_{na} - B_{np})]^2 \right) \quad (6)$$

Where:

N is the number of training images

T_{na} is the actual number of trees in the n^{th} testing image

T_{np} is the predicted number of trees in the n^{th} testing image

B_{na} is the actual aboveground biomass of the n^{th} testing image

B_{np} is the predicted biomass of the n^{th} testing image

S is a scaling function that is calculated at the beginning of the optimization. It applies a gain and offset to the biomass data so that it has the same range as the tree counts.

Relative_importance is the relative importance of the error in the biomass estimate compared to the importance in the error in the tree count estimate. This was set to 0.5 for all runs to help ensure that the program found real trees, since the program may be able to obtain accurate biomass estimates even if it identifies trees improperly.

The second fitness equation (Equation 6) is used for optimizations in which a new biomass equation is derived each time a new set of parameters is evaluated, whereas the first equation (Equation 5) is used in all other cases. These fitness functions minimize only the error in the total number of trees and stand biomass in a training scene, rather than the error in the position of trees or in the crown diameters of trees. This simplification was made because of the considerable time

required to find the exact locations of trees and obtaining accurate measurements of their crown diameters.

3.2.4.1 Optimization techniques

There are a wide range of optimization algorithms available. For this study, only algorithms that could simultaneously optimize multiple variables were considered, since there are either ten (orthophotograph) or nine (LiDAR) parameters controlling the tree finding algorithm. Additionally, using an unconstrained algorithm (i.e., one that does not put fixed bounds on the parameter ranges) will make the training tolerant to user errors in specifying reasonable parameter ranges, hence adding robustness. Unconstrained, multivariate algorithms can be divided into two classes: Local and global optimization techniques. To understand the difference, please refer to Figure 11. Local optimization techniques assume that there is only a single maximum present in the dataset (the parameter space surface is unimodal), and attempt to climb the peak nearest the starting point as efficiently as possible (Charbonneau, 2002). Global optimization techniques conduct a more extensive search, and attempt to find the largest (global) peak in the parameter space (Charbonneau, 2002). For this study, two optimization algorithms have been implemented and their results will be compared. The algorithms are a modified version of the Nelder-Mead simplex local optimization algorithm (Nelder and Mead, 1964) and a genetic algorithm, which is a global optimization technique. These techniques were chosen because they are both commonly used techniques that have been shown by past studies to be highly effective (see Sections 3.2.4.1.1 and 3.2.4.1.2).

3.2.4.1.1 The Nelder-Mead simplex algorithm

The Nelder-Mead simplex algorithm (Nelder and Mead, 1964) is a relatively simple algorithm that makes few assumptions about the data. This algorithm is considered to be very robust by some authors. Charbonneau (2002) refers to the Nelder-Mead simplex as ‘pseudoglobal’ and notes without proof that it is much more robust than derivative-based techniques. Hereford (2001) performed a quantitative comparison of the Nelder-Mead simplex

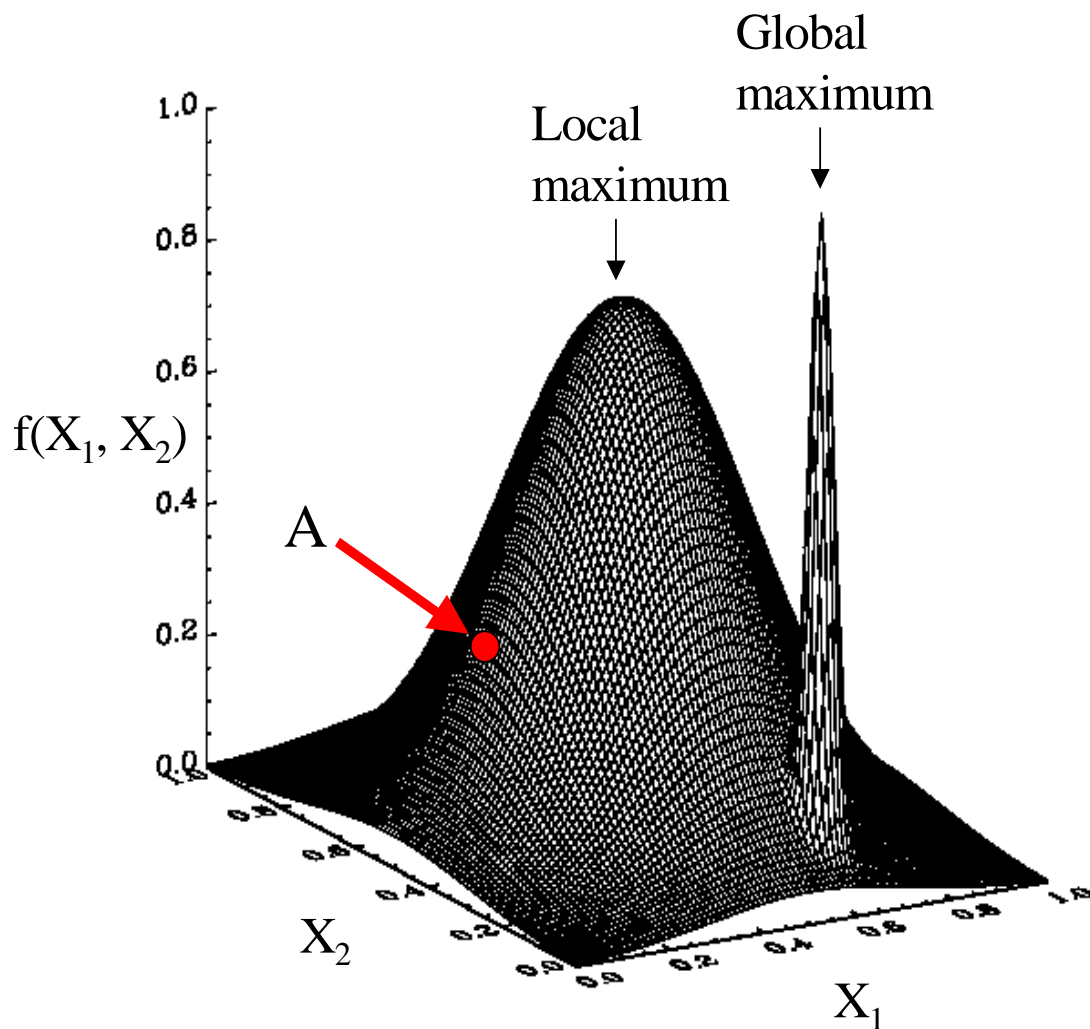


Figure 11. The difference between local and global optimization. This figure shows the value of a function, f , which has two parameters, X_1 and X_2 . In this case, the function is the sum of two two-dimensional Gaussians with different variances. If a local optimization algorithm were started at point A, it would climb the wider of the two Gaussians, thereby finding a local rather than a global maximum. A global optimization technique would attempt to find the global maximum (the peak of the narrower Gaussian). Modified from Charbonneau (2002).

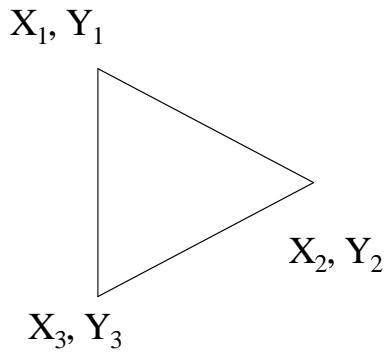


Figure 12. A representation of the simplex used for simultaneously optimizing two parameters.

algorithm, the gradient-based quasi-Newton algorithm, and a genetic algorithm. The author found that the Nelder-Mead simplex algorithm was able to obtain more optimal solutions than the quasi-Newton technique, and that the Nelder-Mead simplex algorithm obtained results that were either as good or nearly as good as those obtained by the genetic algorithm. Despite these plaudits, Wright (1996) warns that the Nelder-Mead simplex algorithm can ‘bog down’ in some situations, making

negligible progress even though the current solution is far from optimal.

The basis of the Nelder-Mead simplex algorithm is a simple geometric figure, known as a simplex (see Figure 12), that crawls across feature space in a controlled way seeking an optimal solution. The simplex has $N+1$ vertices (N = the number of parameters). The coordinates of these vertices are based on starting parameters specified by the user (\mathbf{X}_1), and N other coordinates based on \mathbf{X}_i that are set as follows (Kuester and Mize, 1973, p. 298-317):

$$\mathbf{X}_j = \mathbf{X}_1 + \xi_j, j = 2, 3, \dots, N+1 \quad (7)$$

ξ_j is determined using Table 5.

Table 5. System for determining ξ_j , which is used for determining the starting vertices for the simplex. In this table:

j	$\xi_{1,j}$	$\xi_{2,j}$	•	$\xi_{N-1,j}$	$\xi_{N,j}$
2	p	q	•	q	q
3	q	p	•	q	q
•	•	•	•	•	•
N	q	q	•	p	q
N+1	q	q	•	q	p

In Table 5, the values of p and q are determined using these equations:

$$p = \frac{a}{N\sqrt{2}}(\sqrt{N+1} + N - 1) \quad (8)$$

$$q = \frac{a}{N\sqrt{2}}(\sqrt{N+1} - 1) \quad (9)$$

Where:

a is the lengths of the sides of the simplex.

In most implementations of this algorithm, the value of a is set to a single, fixed value. However, in this case the parameters have very different reasonable ranges. As such, the values of a were set to different values for each parameter (P_i , $i = 1, 2, \dots, N$) based on the maximum ($P_{\max, i}$) and minimum ($P_{\min, i}$) values that are considered reasonable for the parameter.

$$a_i = P_{\max, i} - P_{\min, i} \quad (10)$$

After formation, a score is determined for each of the simplex's vertices using either Equation 5 or Equation 6.

The simplex then moves across the parameter space in a controlled way seeking the parameters that give the lowest score. This is accomplished by either replacing the worst point by one of three types of new point (a reflection, contraction, or expansion point), or by moving all points incrementally towards the best point. The procedure for doing this is described below and depicted diagrammatically in Figure 13 (after Kuester and Mize, 1973, p. 298-317; Wright, 1996):

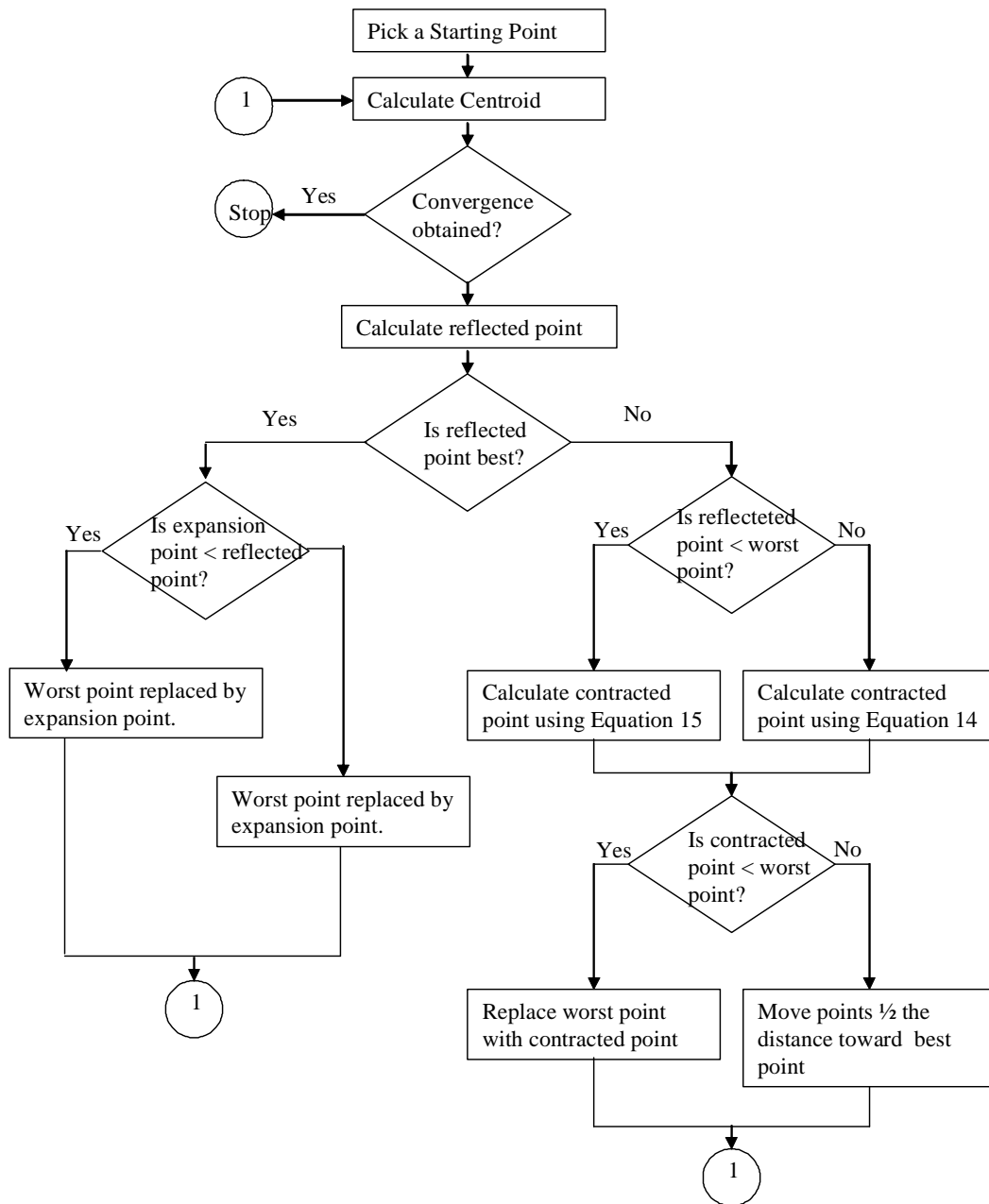


Figure 13. A flow chart showing the Nelder-Mead simplex algorithm. This figure is based on Figure 9.1 in Kuester and Mize (1973).

1. Determine the parameters for a reflected point, $X(\text{reflected})$, and determine its score. A reflected point can be loosely considered to be the mirror image of the worst point (projected through the centroid) and is calculated using the equation:

$$X_{i,j}(\text{reflected}) = \bar{X}_{i,c} + \alpha(\bar{X}_{i,c} - X_{i,j}(\text{worst})) \quad (11)$$

Where:

$X(\text{worst})$ are the parameters of the coordinates with the worst score

α is a constant which was set to 1.0

$\bar{X}_{i,c}$ is the centroid for all but the worst point, and is calculated using the equation:

$$\bar{X}_{i,c} = \frac{1}{K-1} \left[\sum_{j=1}^K X_{i,j} - X_{i,j}(\text{worst}) \right] \quad (12)$$

Where:

$$K = N+1 \quad (13)$$

2. If the reflected point calculated in step 1 has the worst score, calculate the parameters for a contraction point, $X(\text{contracted})$ using the equation:

$$X_{i,j}(\text{contracted}) = \bar{X}_{i,c} - \beta(\bar{X}_{i,c} - X_{i,j}(\text{reflected})) \quad (14)$$

Where:

β is a constant that was set to 0.5

If the reflected point calculated in step 1 had neither the best nor worst score, the reflected point is used in place of the worst point in the previous equation:

$$X_{i,j}(\text{contracted}) = \bar{X}_{i,c} - \beta(\bar{X}_{i,c} - X_{i,j}(\text{worst})) \quad (15)$$

If the contraction point calculated using either of the two previous equations has a better score than the worst point score, then the contraction point replaces the worst point. However, if the contraction point does not have a better score, all the points except the best point are moved incrementally towards the point with the best score, $X(\text{best})$ to form new points, $X(\text{new})$:

$$X_{i,j}(\text{new}) = \frac{(X_{i,j}(\text{best}) + X_{i,j}(\text{old}))}{2} \quad (16)$$

1. If the reflected point calculated in step 1 had the best score, an expansion point, $X(\text{expansion})$, is created, which is essentially a reflected point that extends further away from the centroid and the worst point than the reflected point calculated in Equation 11. It is calculated using the equation:

$$X_{i,j}(\text{expansion}) = \bar{X}_{i,c} + \gamma(X_{i,j}(\text{reflected}) - \bar{X}_{i,c}) \quad (17)$$

Where:

γ is a constant that was set to 2.0

If the expansion point is better than the reflected point, then the expansion point replaces the worst point. Otherwise the reflected point replaces the worst point.

This process is repeated until a stopping criterion is met. For this implementation, the process was stopped when all the vertices had identical scores or if the lowest score remained unchanged for 100 iterations. However, Press *et al.* (1992, p. 408-412) recommend restarting the optimization after it has been stopped using the convergence point as the starting point. In this

implementation, the algorithm was restarted until restarting produced no improvement in the lowest score.

In order to generate the starting points, a user specified number of points were generated, with parameters set randomly to values between P_{min} and P_{max} . Of these, the points with the best (lowest) scores were selected as starting points for the Nelder-Mead simplex algorithm. Restarting at multiple points is beneficial because the Nelder-Mead simplex is not a global optimization routine, so restarting at multiple points increases the chances that the global optimum will be found. Additionally, overfitting is problematic in this study. Overfitting occurs when the program learns unimportant details of individual cases (i.e., noise) rather than the underlying data pattern (Masters, 1993, p. 11-12). This is shown diagrammatically in Figure 14. Overfitting generally occurs when a large number of parameters are set using a relatively small amount of training data, which is the case in this study. By having multiple good solutions, it may be possible to minimize the effects of overfitting. This will be discussed in Section 3.6.

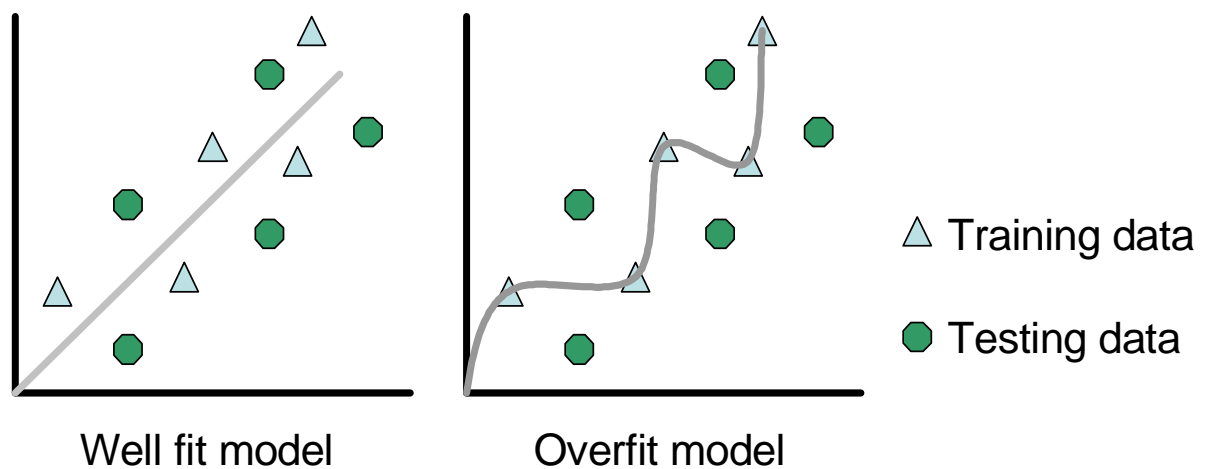


Figure 14. A visual depiction of a well fit model and an overfit model.

3.2.4.1.2 A genetic algorithm

Genetic algorithms are a class of optimization algorithms that gain inspiration from the processes of evolution and genetics. In general, genetic algorithms start by producing a population of individuals that have genes. The genes correspond to parameters for the problem

that is being optimized. Each organism usually has a different set of genes, and the genes, together with a fitness function, determine the fitness of the organism (i.e., how well the genes solve the problem that is being optimized). The algorithm then goes through a process that is similar to evolution: Individuals that are highly fit have a greater chance of passing their genes on to future organisms than to less fit individuals. Therefore, the population progressively moves towards a more optimized state. New offspring are usually produced by two genetically inspired techniques, crossovers and mutations. In crossovers, the genes of two organisms are merged (i.e., the new individual has genes from both parents). Mutations involve setting one or more genes in the new individual to a random value.

In image processing, the use of genetic algorithms is moderately common. Papers include Bhanu *et al.* (1995), who used a genetic algorithm to set the parameters for an algorithm that was used to segment photographs of outdoor scenes; Agbanhan *et al.* (2000), who used a genetic algorithm in conjunction with a geometrical model to find the position and orientation of fish in unprocessed digital images; and Roth and Levine (1994), who used a genetic algorithm to extract geometric primitives from images. Despite their popularity and effectiveness in many applications, few papers have specifically explored their use in remote sensing. Yu *et al.* (2002) used genetic algorithms to select features in hyperspectral imagery prior to classifying the image using the fuzzy k-nearest neighbors techniques, and Stiteler and Hopkins (2000) used genetic algorithms to create templates to locate trees in high resolution images.

The genetic algorithm designed for this study sets the genes to real numbers rather than binary representations of the parameter values. Using real rather than binary representations in a genetic algorithms is a highly controversial subject, and a schism exists between practitioners who have had considerable success using real parameters and some theorists (Goldberg, 1991). Despite the controversy, there are a number of distinct advantages to using real parameters in this implementation. The two main advantages are that genetic algorithms with real parameters tend to converge to a solution more quickly than those based on binary representations, and it is possible to maintain greater precision using real numbers (Goldberg, 1991; Wright, 1991). These are both important, since the tree finding algorithm is very computationally expensive, and it has been observed to be sensitive to small changes in parameter values. In his paper, Goldberg (1991)

developed a theory to explain why real representations work so well. Unfortunately, one of the conclusions of the theory is that in some cases solutions can be blocked, therefore causing the algorithm to miss the global optimum, which suggests that the performance of the genetic algorithm may not provide a perfect test of whether a global optimization procedure is useful in this study.

The first step in implementing the algorithm is to initialize a population of organisms. The number of organisms is specified by the user, and the organisms have their genes set to random values taken from uniform distributions. The bounds of these distributions are determined by the user based on what he or she considers to be the reasonable upper and lower limits of each parameter. Each organism is randomly assigned an origin (uniform crossover, averaging crossover or mutation), and its fitness is assessed by running the computer vision program using its genes.

Children are then created and evaluated one at a time. The steps for doing this are:

1. Select the parents. This is done using a roulette system, which chooses parents randomly based on fitness. To assign the probability of selection to each organism in the population, the following equation was developed for this study:

$$p(n) = \frac{\frac{1}{\sqrt{F_n + 1}}}{\sum_{i=1}^P \frac{1}{\sqrt{F_i + 1}}} \quad (18)$$

Where:

$p(n)$ is the probability that the n^{th} organism will be selected

P is the population size

F is the fitness

In this equation organisms that are fit (F is small), have a greater probability of selection. The square root terms were included in order to be conservative, since it slows the decrease in

selection probability that occurs with decreasing fitness, thereby helping to ensure that the population remains genetically diverse. Plus one terms prevent the possibility of division by zero errors.

2. Choose an origin for the child. This is done by examining the population and seeing how many of the organisms in the population arose from averaging and uniform crossovers (see step 3 below) and how many arose from mutations. These numbers are used to determine the probabilities associated with each reproduction type, and the choice of origins is made randomly based upon these probabilities. This approach has been shown to be effective in the past (Davis, 1991). In order to prevent a reproduction type from becoming extinct, in this implementation each probability was constrained to not fall below 0.1.
3. If the origin of the new child is determined to be a uniform crossover, a uniform crossover (Davis, 1991) is performed by setting each of the child's genes to that of one of the two parents, the contributing parent being chosen randomly for each gene. In order to prevent the creation of children with the same genetic composition as their parents, the algorithm used in this study makes sure that the child is genetically different from both parents. If the child is identical to one of the parents, a mutation takes place.
4. Averaging crossovers, which Davis (1991) found to be highly effective when real parameter coding is used, are performed by setting each child's genes to the average of those of its two parents.
5. Mutations are performed by setting the child's genes equal to those of one of its parents (chosen at random). A single gene is then selected for mutation. The value of the new gene is set to the value of the old gene plus a random number selected from a normal distribution. The distribution is centered at zero, and has a standard deviation equal to the mutation strength (set by the user) times the difference between the maximum and minimum parameter values specified by the user.
6. Evaluate the child's fitness using Equation 3 or 4. If the child is more fit than the weakest member of the population, the weakest member is replaced with the child. Otherwise the child is abandoned.

This process of creating and evaluating children is repeated a user-specified number of times.

3.3 Study site, available data, and data preprocessing

To perform a first test of the algorithm, data from the loblolly pine plantations in a 4.6 km² section of the Appomattox-Buckingham State Forest (ABSF) bound by coordinates (37.43409N, 78.69357W), (37.40450N, 78.65787W) were used. The ABSF is located in the Piedmont physiographic province of central Virginia (Figure 15), and the forests in the ABSF study area

Location of the Appomattox-Buckingham State Forest

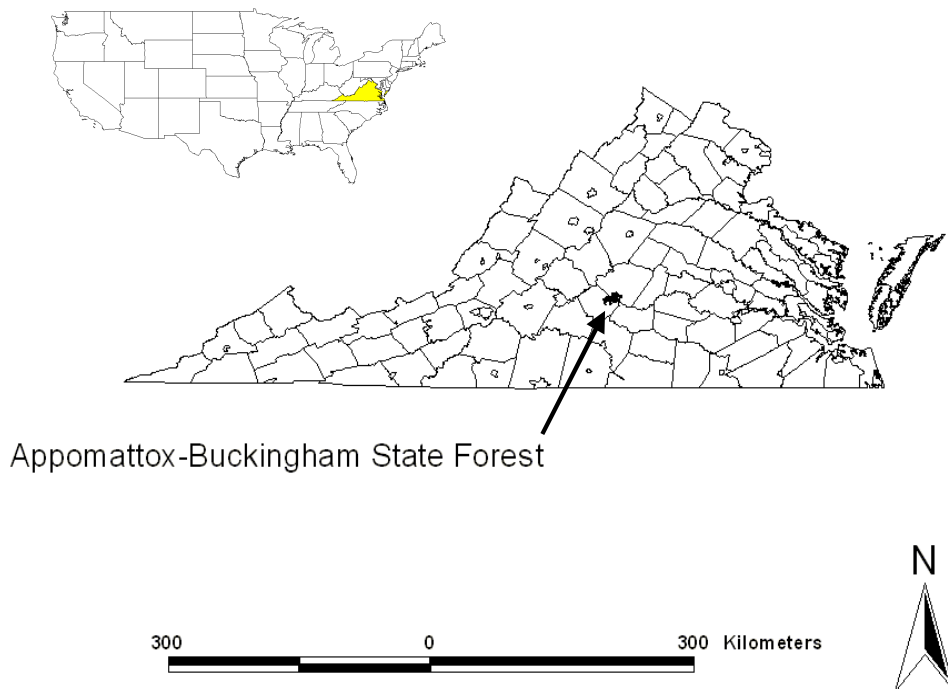


Figure 15. A map showing the location of the Appomattox-Buckingham State Forest

consist of both natural hardwood and extensively managed loblolly pine planted at different densities. Although the loblolly pine stands in the ABSF range in age from three to 65 years old, for this study, only stands between eleven and sixteen years of age (seven to thirteen at the time of the orthophotograph acquisition) were examined. This age range represents 21% of the loblolly pine present in the study area. The decision to examine this narrow age range was made because younger stands contained trees that were too small to be identified in the imagery used for this study, and older stands contained large numbers of subdominant trees with crowns that were not visible in the orthophotograph.

The imagery used for this analysis consisted of a false color orthophotograph with a 0.5 m spatial resolution and small footprint LiDAR data. The orthophotograph was created using aerial photographs acquired by NASA during the fall of 1999 at a scale of 1:13,800 and digitized by EarthData International. The LiDAR data were collected by 3Di LLC in 2002 using a Digital Airborne Topographic Imaging System II (DATIS II) sensor. The sensor has a one meter posting interval, and can collect up to five returns per pulse. Prior to shipping the data, 3Di separated the last returns into ground and non-ground hits. For this dissertation, a CHM was required. This was produced using the Kriging technique to interpolate surfaces from the first LiDAR return and the ground hits, and then subtracting the two surfaces. The interpolated surfaces had a spatial resolution of 0.5m. The Kriging technique was selected because Popescu *et al.* (2002) found that it produces more accurate results than the other interpolation techniques they evaluated.

Although the orthophotograph used in the analysis is three years older than the ground data measurements, this does not pose a large problem. The reason is that these data are only used to determine stand density, and in the field it was observed that there were a negligible number of snags, indicating that attrition due to mortality is very low. Therefore it is likely that the density of trees remained relatively unchanged during this interval.

In addition to the imagery, a forest cover map of the forest was produced in 1999 by the Virginia Department of Forestry. This map includes the boundaries of stands, and information on the species found in the stand and the age of the stand. Since the boundaries of several of the stands were imprecisely digitized, the boundaries of stands of interest were redigitized based on the orthophotograph.

3.4 Ground data collection

Ground data for the analysis consist of 25 fixed radius (radius = 15m) plots placed in eleven to sixteen year old pine plantations in the study area (Tables 6 and 7, Figure 16). Plot centers were measured using a Corvallis Microtechnology (CMT) March II GPS (Corvallis Microtechnology, Inc., Corvallis, OR) unit with the internal antenna. Data were collected for ninety seconds, although in some cases data from the entire period were not usable. After collection, data were postprocessed using National Geodetic Survey Continuously Operating Reference Stations (CORS) data from either Richmond, Blacksburg, or Fan Mountain, Virginia together with the PC-GPS software package (Corvallis Microtechnology, Inc., Corvallis, OR). The manufacturer of the unit indicates that postprocessed data have an accuracy of 1.5 - 2.5m (CMT, 2004), although the actual error may be higher for some plots due to multipath errors and the small amount of useable GPS data obtained at some plots.

In all plots, the number of trees with $DBH \geq 7$ cm were counted. The seven centimeter cutoff was selected because it was visually observed on the ground that in the plots measured for the study, nearly none of the trees with a $DBH < 7$ cm were overstory trees, and hence would generally not be visible in the remotely sensed images. Plots 1 - 11 were measured during the summer of 2002. For these plots, the DBH of ten percent of the trees were measured at the time of the initial survey, and the biomass equation for dominant loblolly pine that was developed by Naidu *et al.* (1998) (see discussion in Section 2.3.2) was used to obtain initial biomass estimates. These initial biomass estimates contained large degrees of uncertainty due to the unusually high heterogeneity of the plots. Therefore the number of measurements needed to obtain an

Table 6. Attributes of the plots used for the study. Sampled trees refers to the number of trees for which the DBH was recorded. The confidence intervals surrounding the biomass estimates are 95% intervals.

Plot	Age (2003)	Density, DBH \geq 7 cm (trees / ha)	Estimated density, DBH \geq 10 cm (trees / ha)	Photo interpreted density (trees / ha)	Sampled trees	Aboveground biomass, DBH \geq 7cm (t / ha)	Aboveground biomass, DBH \geq 10 cm (t / ha)
1	13	1457	1132	1316	49	69.7 \pm 9.2	64.8 \pm 7.5
2	15	1302	1146	1528	56	85.2 \pm 7.6	82.9 \pm 6.5
3	16	1684	1528	1698	44	137.9 \pm 15.1	136.0 \pm 12.3
4	16	1358	1259	1712	58	77.4 \pm 7.5	77.7 \pm 5.4
5	11	1273	1188	1542	56	76.0 \pm 5.4	74.4 \pm 4.8
6	13	1740	1302	1570	71	81.9 \pm 8.5	75.1 \pm 6.9
7	13	1471	1401	1627	46	88.4 \pm 6.7	87.3 \pm 6.1
8	13	2037	1882	2136	27	95.8 \pm 11.5	92.9 \pm 9.8
9	13	1500	1118	1684	67	67.1 \pm 5.5	61.1 \pm 4.0
10	13	2009	1966	2306	51	103.4 \pm 8.4	103.0 \pm 7.9
11	11	1952	1839	1938	33	94.4 \pm 10.3	92.1 \pm 9.8
12	13	1924	1896	2080	60	87.6 \pm 5.2	87.1 \pm 5.0
13	13	1924	1853	2193	60	105.6 \pm 7.6	105.4 \pm 6.9
14	11	1401	1302	1613	60	87.9 \pm 5.8	86.4 \pm 5.0
15	16	2264	1627	1485	60	113.7 \pm 16.0	104.8 \pm 11.9
16	16	1839	1415	1556	60	125.3 \pm 14.9	118.7 \pm 10.9
17	17	1712	1542	1896	60	115.3 \pm 10.8	112.6 \pm 9.3
18	12	1740	1330	2009	60	75.4 \pm 8.9	70.2 \pm 5.8
19	15	1924	1768	1768	60	117.0 \pm 10.9	114.7 \pm 9.6
20	15	1952	1825	1655	60	117.0 \pm 11.3	115.2 \pm 10.3
21	13	1401	1330	1401	60	97.2 \pm 8.2	96.2 \pm 7.6
22	17	1457	1316	1995	60	114.5 \pm 9.9	112.5 \pm 8.3
23	17	1485	1415	1924	60	68.3 \pm 5.0	67.2 \pm 4.7
24	11	2179	1853	1924	60	118.3 \pm 12.2	113.5 \pm 10.0
25	11	1556	1302	1698	60	76.5 \pm 6.1	73.4 \pm 4.4
Mean	14	1698	1500	1768	56	95.9	92.9

Table 7. Coordinates of plots used for the study.

Plot	Latitude	Longitude
1	37.42728	-78.69213
2	37.42886	-78.69218
3	37.42854	-78.67583
4	37.42721	-78.67156
5	37.42601	-78.67083
6	37.42858	-78.66806
7	37.42863	-78.66626
8	37.42811	-78.66700
9	37.42805	-78.66636
10	37.42484	-78.66493
11	37.42705	-78.66521
12	37.42762	-78.66534
13	37.42841	-78.66529
14	37.42693	-78.67102
15	37.42876	-78.67479
16	37.43026	-78.67577
17	37.43300	-78.67780
18	37.43141	-78.69262
19	37.42952	-78.68963
20	37.42945	-78.69286
21	37.43577	-78.69285
22	37.41178	-78.67125
23	37.40900	-78.67681
24	37.40791	-78.68412
25	37.40597	-78.68516

Plot Locations

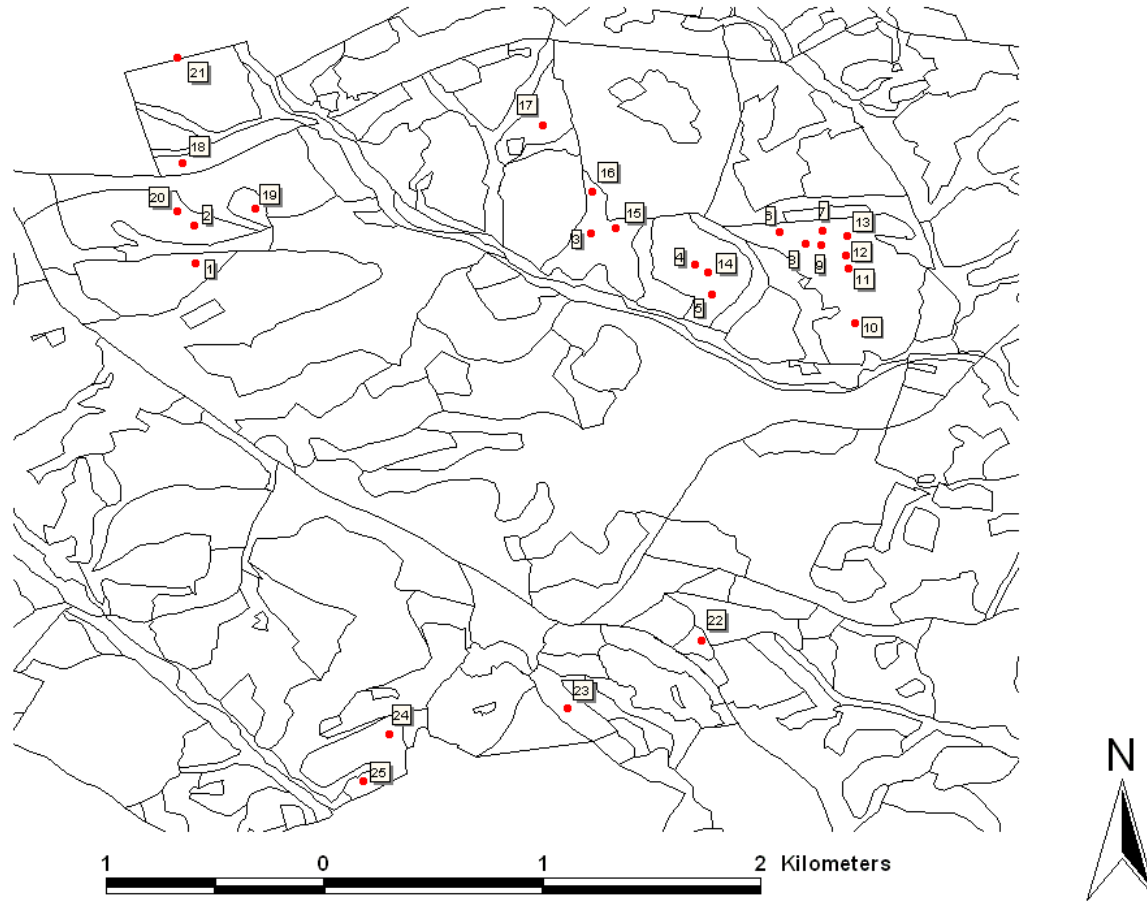


Figure 16. Map showing the locations of the study plots overlain on a stand map of the Appomattox-Buckingham State Forest.

allowable biomass error of 10% in each plot was calculated using the following equation (Shiver and Borders, 1996; p. 47):

$$n = \frac{4N(CV)^2}{(AE)^2 N + 4(CV)^2} \quad (19)$$

Where:

n is the number of samples required

N is the number of trees in the plot

CV is the coefficient of variation of the biomass measurements

AE is the allowable error in biomass expressed as a percentage

The plots were resurveyed, and the DBH and dominance status was determined for at least $n + 0.1(n)$ trees.

Plots 12 - 25 were measured in the spring of 2003. In order to make the data collection more efficient, it was decided to visit each stand only once. Therefore the diameters and dominance statuses of sixty trees were obtained. The number sixty was chosen based on the biomass uncertainty associated with plots 1 - 11, since this number is a conservative estimate of the average number of trees needed to achieve an uncertainty of approximately 10% (Figure 17). For all plots, trees were considered to be suppressed if the tops of their crowns were below the level of their neighbors' crowns, and were considered to be dominant if their crowns were at or above the level of their neighbors. The number of trees with $DBH \geq 10\text{cm}$ was calculated by multiplying the number of trees in each plot by the fraction of the measured trees in each plot which had $DBH \geq 10\text{cm}$.

3.5 Biomass estimation

Biomass is a very important forest parameter, but obtaining direct measurements of biomass is often impractical since it requires that trees be destructively sampled and weighed. As a result, many researchers have developed equations that predict the biomass of different tree components based upon values that can be rapidly and non-destructively obtained. In most cases,

the biomass samples used in constructing the equations are dried prior to weighing in order to make the equations insensitive to temporal and spatial differences in moisture content.

Many researchers have developed equations to predict the biomass of different tree components (Table 8). Although these equations take several forms, and utilize different measurements to make predictions, the r^2 values given in Table 8 suggest that good results can be readily obtained for a wide variety of loblolly pine stands. For this study, the total above ground biomass equations developed by Naidu *et al.* (1998) were used. This choice was made because: 1) the equations were derived for a Piedmont site; 2) the age range of the stands used to derive the equations overlaps the ages of the stands in this study; and 3) the equations only require that the stem diameter and dominance status be recorded, thereby allowing rapid data collection.

3.6 Evaluating the algorithm

The algorithm was evaluated by 1) training and testing using the entire dataset, and 2) training and testing using independent subsets of the data. In the first case, using the same data for testing and training will give artificially high results due to overfitting (see Section 3.2.4.1.1 and Masters, 1993, p. 11 - 12). However, it does provide a reasonable upper limit to the expected accuracy values, since sampling effects (choosing more or less fortuitous training and testing sets) do not exist. It also helps identify which plots are clear outliers.

For the runs in which subsets of plots were used for training and testing, the 25 ground plots were divided into twelve training plots and thirteen testing plots. There was no overlap between the testing and training plots. For this study, two sets of training and testing data were randomly selected from the 25 ground plots. The training data were then further subdivided randomly into sets of six and three plots, where the six plot sets are subsets of the twelve plot sets, and the three plot sets are subsets of the six plot sets (see Table 9). This was done because it was observed that the data contain outliers. Therefore if subsets were not used (e.g., the sets of three or six points were selected randomly from the population of plots) it would not be possible to tell whether differences in accuracy were the result of the quantity of training data used in the optimization or of choosing fortuitous points for training and testing. For example, assume an overall pattern exists whereby

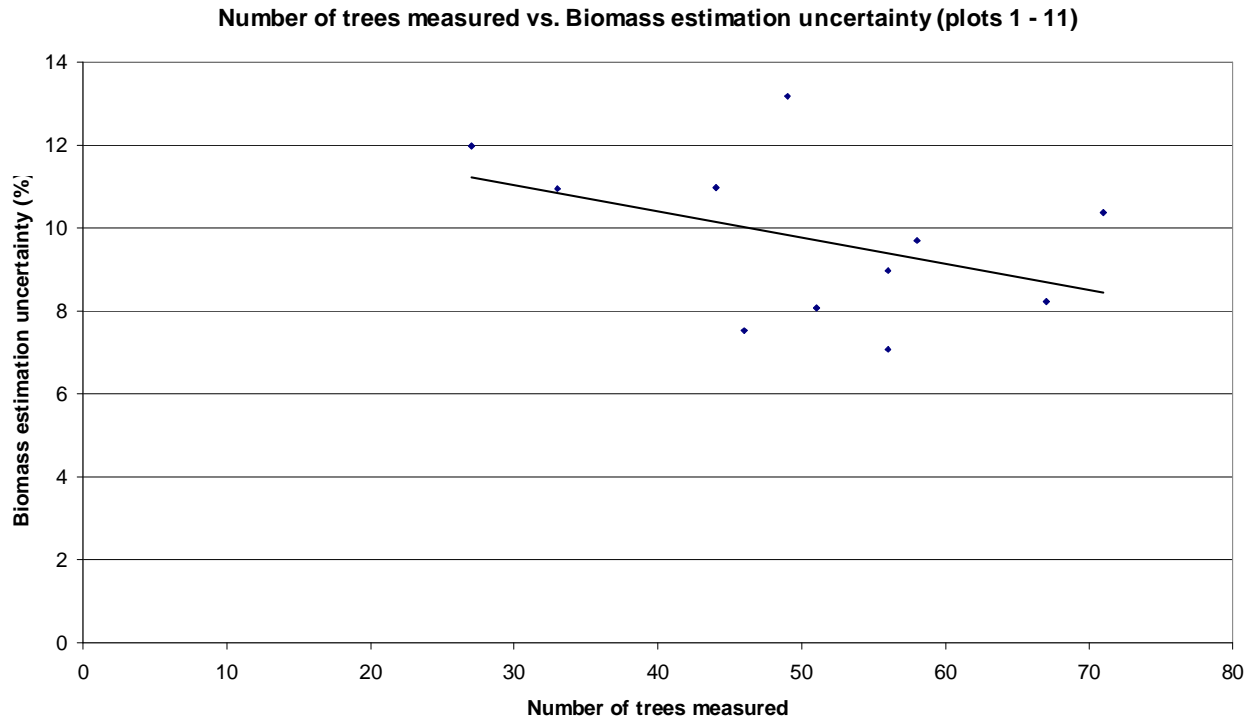


Figure 17. The relationship between the number of trees in plots 1 - 11 for which DBH and dominance status were measured, and the uncertainty in the biomass estimate for trees with DBH ≥ 7 cm. It indicates that measuring the DBH and dominance status of sixty trees per plot is a conservative estimate of the number of trees that need to be measured to achieve an uncertainty of 10%.

Table 8. Survey of equations developed for predicting the biomass of aboveground loblolly pine components. A = Age, C = Relative branch height, D = DBH, L = Crown length, T = Density.

Paper	Component	Form	r ²	Location	Forest type
Baldwin <i>et al.</i> (1997)	Branch	$\ln(B)=b_0 + b_1[\ln(D)]+b_2[\ln(L)]+b_3[\ln(R)]$	0.93	Louisiana, Virginia, North Carolina	Loblolly pine (age 9 to 41)
Baldwin <i>et al.</i> (1997)	New foliage	$\ln(B)=b_0 + b_1[\ln(D)]+b_2[\ln(L)]+b_3[\ln(R)]$	0.88	Louisiana, Virginia, North Carolina	Loblolly pine (age 9 to 41)
Baldwin <i>et al.</i> (1997)	Old foliage	$\ln(B)=b_0 + b_1[\ln(D)]+b_2[\ln(R)]$	0.88	Louisiana, Virginia, North Carolina	Loblolly pine (age 9 to 41)
Clark (III) and Taras (1976)	Total above ground	$\log(B)=b_0 + b_1[\log(D^2H)]$	0.99	Central Alabama	Non-plantation loblolly pine (age 37 to 47)
Edwards and McNab (1979)	Total above ground	$\log(B)=b_0 + b_1[\log(D^2H)]$	0.99	Upper Coastal Plain, Georgia	Loblolly pine (age 2 to 8)
Hepp and Brister (1982)	Total crown	$\ln(B)=b_0 + b_1[\ln(D)]+b_2[\ln(R)]$	0.9377	Lower Coastal Plain, Carolinas	Loblolly pine (age 10 to 27)
Hepp and Brister (1982)	Total crown	$\ln(B) = b_0 + b_1[(\ln(C)) + b_2[\ln(A)] + b_3[\ln(T)]$	0.866	Lower Coastal Plain, Carolinas	Loblolly pine (age 10 to 27)
Hepp and Brister (1982)	Branch	$\ln(B)=b_0 + b_1[\ln(D)]+b_2[\ln(R)]$	0.9385	Lower Coastal Plain, Carolinas	Loblolly pine (age 10 to 27)
Hepp and Brister (1982)	Branch	$\ln(B) = b_0 + b_1[(\ln(D)) + b_2[\ln(A)] + b_3[\ln(T)]$		Lower Coastal Plain, Carolinas	Loblolly pine (age 10 to 27)
Naidu <i>et al.</i> (1998)	Bole	$\log(B)=b_0 + b_1[\log(D)]$	0.98	Piedmont region, North Carolina	Dominant loblolly pine (age 10 to 48)
Naidu <i>et al.</i> (1998)	Bole	$\log(B)=b_0 + b_1[\log(D)]$	0.98	Piedmont region, North Carolina	Suppressed loblolly pine (age 10 to 48)
Naidu <i>et al.</i> (1998)	Needle	$\log(B)=b_0 + b_1[\log(D)]$	0.92	Piedmont region, North Carolina	Dominant loblolly pine (age 10 to 48)
Naidu <i>et al.</i> (1998)	Needle	$\log(B)=b_0 + b_1[\log(D)]$	0.85	Piedmont region, North Carolina	Suppressed loblolly pine (age 10 to 48)
Naidu <i>et al.</i> (1998)	Branch	$\log(B)=b_0 + b_1[\log(D)]$	0.92	Piedmont region, North Carolina	Dominant loblolly pine (age 10 to 48)
Naidu <i>et al.</i> (1998)	Branch	$\log(B)=b_0 + b_1[\log(D)]$	0.86	Piedmont region, North Carolina	Suppressed loblolly pine (age 10 to 48)

Paper	Component	Form	r ²	Location	Forest type
Naidu <i>et al.</i> (1998)	Total above ground	$\log(B)=b_0 + b_1[\log(D)]$	0.99	Piedmont region, North Carolina	Dominant loblolly pine (age 10 to 48)
Naidu <i>et al.</i> (1998)	Total above ground	$\log(B)=b_0 + b_1[\log(D)]$	0.98	Piedmont region, North Carolina	Suppressed loblolly pine (age 10 to 48)
Nemeth (1973)	Total above ground	$\ln(B)=b_0 + b_1[\ln(D+1)]+b_2[\ln(D+1)]^2+b_3[\ln(L)]$	0.985	Coastal Plain region, North Carolina	Slash pine (age 4 to 8) and Loblolly pine (age 4 to 12)
Nemeth (1973)	Main stem	$\ln(B)=b_0 + b_1[\ln(D+1)]+b_2[\ln(D+1)]^2+b_3[\ln(H)]$	0.991	Coastal Plain region, North Carolina	Slash pine (age 4 to 8) and loblolly pine (age 4 to 12)
Nemeth (1973)	Total branch	$\ln(B)=b_0 + b_1[\ln(D+1)]+b_2[\ln(D+1)]^2+b_3[\ln(L)]+b_4[\ln(H)]+b_5[\ln(H)]^2$	0.947	Coastal Plain region, North Carolina	Slash pine (age 4 to 8) and loblolly pine (age 4 to 12)
Pehl <i>et al.</i> (1984)	Foliage	$\ln(B)= b_0 + b_1[\ln(D)]$	0.83	Coastal Plain region, Texas	Loblolly pine (age 25)
Pehl <i>et al.</i> (1984)	Live branch	$\ln(B)= b_0 + b_1[\ln(D)]$	0.91	Coastal Plain region, Texas	Loblolly pine (age 25)
Pehl <i>et al.</i> (1984)	Dead branch	$B = b_0 + b_1(D^2)$	0.62	Coastal Plain region, Texas	Loblolly pine (age 25)
Pehl <i>et al.</i> (1984)	Stem bark	$B = b_0 + b_1(D^2)$	0.88	Coastal Plain region, Texas	Loblolly pine (age 25)
Pehl <i>et al.</i> (1984)	Stem wood	$B = b_0 + b_1(D^2)$	0.99	Coastal Plain region, Texas	Loblolly pine (age 25)
Van Lear <i>et al.</i> (1984)	Total stem (5cm top)	$\ln(B)=b_0 + b_1[\log(D)]$	0.985	Piedmont region, South Carolina	Loblolly pine (age 41)
Van Lear <i>et al.</i> (1984)	Branches	$\ln(B)=b_0 + b_1[\log(D)]$	0.953	Piedmont region, South Carolina	Loblolly pine (age 41)
Van Lear <i>et al.</i> (1984)	Foliage	$\ln(B)=b_0 + b_1[\log(D)]$	0.937	Piedmont region, South Carolina	Loblolly pine (age 41)
Van Lear <i>et al.</i> (1984)	Total above ground	$\ln(B)=b_0 + b_1[\log(D)]$	0.985	Piedmont region, South Carolina	Loblolly pine (age 41)

Table 9. The plots that were selected as testing data (T), training data used in the optimization (Y), and training data not used in optimization (E). The number in parentheses refers to the number of points using during optimization.

Plot	Set 1 (3)	Set 1 (6)	Set 1 (12)	Set 2 (3)	Set 2 (6)	Set 2 (12)
1	T	T	T	Y	Y	Y
2	T	T	T	E	E	Y
3	T	T	T	T	T	T
4	E	E	Y	T	T	T
5	T	T	T	E	E	Y
6	E	E	Y	T	T	T
7	E	Y	Y	T	T	T
8	Y	Y	Y	E	E	Y
9	T	T	T	T	T	T
10	Y	Y	Y	E	E	Y
11	T	T	T	E	Y	Y
12	T	T	T	T	T	T
13	E	Y	Y	T	T	T
14	T	T	T	T	T	T
15	E	E	Y	T	T	T
16	E	Y	Y	T	T	T
17	T	T	T	E	Y	Y
18	T	T	T	E	E	Y
19	E	E	Y	T	T	T
20	E	E	Y	T	T	T
21	Y	Y	Y	Y	Y	Y
22	E	E	Y	T	T	T
23	T	T	T	E	E	Y
24	T	T	T	E	Y	Y
25	T	T	T	Y	Y	Y

using six training data points produces more accurate results than using three training data points. This pattern would in all likelihood be obscured (or even reversed) if the algorithm is trained using six outlying datapoints, and these results are compared to results obtained by training using three non-outliers. Using subsets and the same testing dataset makes it less likely that the three and six member training and testing sets will contain vastly different numbers of outliers. Table 9 shows the plots that were selected.

Using these training data, several tests were performed using the Nelder-Mead simplex optimization algorithm. For this algorithm, fifty points were randomly generated, and the best five were used as starting points. Runs were conducted in which the number of trees and biomass were predicted using different training data. Table 10 shows the runs that were performed. For the test in which the orthophoto counts and LiDAR heights were used to estimate tree count,

Table 10. Tests performed for evaluating the algorithm.

Predicted variable	Imagery used	Training data: Photo interpreted values	Training data: Ground values (DBH > 7 cm)	Training data: Ground values (DBH > 10 cm)
Tree count	Air photos, LiDAR directly	Yes	Yes	Yes
Tree count	Air photo counts together with LiDAR heights	No	No	Yes
Biomass (equation derived during optimization)	LiDAR only	No	No	Yes
Biomass (equation derived after optimization)	LiDAR only, LiDAR heights and air photo tree counts	No	No	Yes

stepwise multiple regression was used ($p \leq 0.05$ to enter, $p \leq 0.1$ to remain) to create the equation using data from the most successful run (based on r and RMSE) in which 25 plots were used for training and testing. Two equation forms were evaluated. For the first form, the actual tree count was the dependent variable, and independent variables were selected from those listed in the left column of Table 4 by the stepwise regression procedure. For the second form, the natural log of the actual tree count was the dependent variable, and independent variables were selected from those listed in the right column of Table 4. After the tests listed in Table 10 have been performed, the results were evaluated using either all 25 plots (in cases where all 25 plots were used for training) or the thirteen testing plots in cases where a subset of the data was used for training. In each case the correlation coefficient (r) and RMSE were calculated.

An attempt was made to find an effective technique for postprocessing the results obtained using the five parameter sets returned by the Nelder-Mead simplex algorithm. The post processing attempted to find a better means of obtaining predictions for the testing stands than just randomly choosing one of the five possible solutions. Post processing was conducted using only the training data, since in a real world situation the practitioner would not have access to the testing data. In cases where only three or six plots were used in the optimization process, the remaining training plots were used in this step. Six post processing methods were evaluated and are described in Table 11.

Table 11. Postprocessing techniques that were evaluated

Abbreviation	Technique
CORv	Calculate the correlation coefficients (r) for the five parameter sets based on the plots used in the optimization (Y) and the training plots that were not used in the optimization (E). Choose the parameter set with the highest r .
CORvy	Calculate the correlation coefficients (r) for the five parameter sets based only on the training plots that were not used in the optimization (E). Choose the parameter set with the highest r .
RMSEv	Calculate the RMSE for the five parameter sets based on the plots used in the optimization (Y) and the training plots that were not used in the optimization (E). Choose the parameter set with the lowest RMSE.
RMSEvy	Calculate the RMSE for the five parameter sets based only on the training plots that were not used in the optimization (E). Choose the parameter set with the lowest RMSE.
RMSEy	Calculate the RMSE for the five parameter sets based only on the training plots that were used for optimization (Y). Choose the parameter set with the lowest RMSE.
Av5	Average the tree counts or biomass values obtained using all five parameter sets, and use the average values as the predictions for the testing plots.

Determining which technique works best was done qualitatively and quantitatively using the testing data. Qualitatively, the correlation coefficients and RMSE values obtained using each method were compared with the average correlation coefficients and RMSE values for the five parameter sets. Quantitatively, the differences between the correlation coefficient or RMSE value obtained using the different postprocessing techniques and the average of all five correlation coefficients or RMSE values for each run were computed. These difference values were then averaged for the runs in which three or six plots were used for training (data for runs in which twelve plots were used were not included since not all post processing techniques were applicable in these runs). The averages for each postprocessing technique were then compared, and Tukey's W procedure (Ott, 1993, p. 818-822) was used to determine whether the differences in the averages for the different techniques were significant at the 95% level.

In addition to determining the number of trees and biomasses of the plots, it is also desirable that the trees that are located by the program correspond to actual trees. To evaluate this, the locations of trees found by a human interpreter were compared with the results from a run in which the number of trees found by a human interpreter was used as training data. This

was done using the most successful (highest r , lowest RMSE) results when all 25 plots and the human tree counts were used as training data. Eight points were then randomly placed in the plot portion of each photo. For the first four points, the nearest tree located by the computer was found, and it was noted whether it overlapped a tree found by the human interpreter. For the second four points, the nearest tree found by the human interpreter was located, and it was noted whether it overlapped a tree found by the computer. Trees found from the human interpreter and the program were considered to correspond if over 50% of the center pixel of the tree found by the program lies within the boundary of a tree found by a human interpreter. Using this information, two calculations were made: The percentage of trees found by the human interpreter that correspond to trees found by the program, and the percentage of trees found by the program that correspond to trees found by the program.

The main evaluation was conducted using the Nelder-Mead simplex algorithm for optimization. However, because there are a large number of parameters and the optimization problem is relatively complex, the problem may benefit from a global optimization routine. To evaluate this, a comparison was conducted between the Nelder-Mead simplex algorithm and the genetic algorithm. The comparison consisted of using both algorithms to optimize for the number of trees with $DBH \geq 7$ cm in a plot for training sets consisting of 3, 12, and 25 plots. Graphs were constructed showing the score versus the number of times the function was evaluated. The comparison was run twice for each set of training data, and stopped after 5000 evaluations. For the Nelder-Mead simplex algorithm, 100 random starting points were created, and processed in order of fitness (i.e., the most fit of the 100 random points was used as the first starting point). For the genetic algorithm, the population size was 100 organisms.

Chapter 4

Results

4.1 Overview

The main objective of this dissertation was to develop an object-oriented algorithm for measuring forest density and biomass using orthophotographs and small footprint LiDAR CHMs, and to write a program that implements the algorithm. Chapter 3 describes the algorithm that was engineered to meet the main objective, and the programs for implementing the algorithm are given in Appendices I - VI. This chapter gives the results of the first test of the algorithm that were obtained using the procedures described in Section 3.6

4.2 Derivation of equation forms

For this study, two equation forms were derived using stepwise multiple linear regression: The equation for predicting tree counts using orthophoto-derived tree counts and LiDAR-derived heights, and the equation for predicting biomass using LiDAR- and orthophoto-derived tree counts and LiDAR derived heights. In the first case, stepwise multiple regression created an equation that contained no height-related terms. It is therefore appears that for these data, LiDAR height information cannot be used to increase the accuracy of the tree count, and no further work was performed to this end. This is somewhat surprising since McCombs *et al.* (2003) obtained an increase in accuracy with the simultaneous use of LiDAR and high spatial resolution passive optical data.

For predicting biomass from tree count and height information, the equation obtained by stepwise multiple linear regression was:

$$B = \beta_0 + \beta_1(H50)^2 \quad (20)$$

Where:

B is the plot biomass

H50 is the median height for the trees in the plot

β_0 and β_1 are coefficients

It is important to note that no terms with tree count were included in this equation, and the same equation was produced when both the LiDAR- and photo-derived tree counts were used. This was rather surprising, and is likely due to the error associated with the tree counts derived from the imagery. To test this, actual tree counts were substituted for the predicted tree counts. In this case, the resulting equation did include tree count terms, so the error level is the likely reason why no tree count terms are included in Equation 20.

4.3 Analysis of the computer vision algorithm after training using all 25 datasets

Tree counts obtained using the full dataset for testing and training are shown in Table 12, and the results for biomass values are shown in Table 13. The correlation coefficients were not squared because in subsequent analyses, some negative correlations were found and squaring the correlation coefficients would lead to ambiguity. Statistically, nearly all the correlation coefficients are highly significant (significance $\geq 95\%$ based on a one tailed test), with the exception of the prediction of the number of trees with $DBH \geq 7$ cm using the LiDAR CHM.

4.3.1 Residual analysis

Residuals were examined by creating residual plots (Figures 18a - f) and by examining first order autocorrelation after regressing the tree counts or biomass values obtained by the program against the training data (Table 13). Figures 18a - f show that with the exception of residuals when the number of photointerpreted trees were used as training and testing data, the residuals are negatively correlated with increasing biomass or density. Figure 19 contains plots with strong positive, strong negative and weak residuals. The presence of residuals suggests that the program was unable to detect some of the structural factors, such as an unusually sparse or
(continued on page 105)

Table 12. Tree count results obtained when all 25 plots were used for testing and training. One or two asterisks indicate significance at the 95%, and 99% level respectively based on a one tailed test.

Imagery	Predicted data	Min. r	Max. r	Avg. r	Min. RMSE (trees / ha)	Max. RMSE (trees / ha)	Avg. RMSE (trees / ha)
Photo	Photointerpreted tree count	0.91**	0.94**	0.93**	83.5 (4.7%)	107.5 (6.1%)	94.8 (5.4%)
Photo	Trees \geq 7cm	0.61**	0.68**	0.65**	210.8 (12.4%)	229.2 (13.5%)	216.5 (12.7%)
Photo	Trees \geq 10cm	0.76**	0.83**	0.80**	155.6 (10.3%)	181.1 (12.2%)	168.4 (11.2%)
LiDAR	Trees \geq 7cm (biomass not used in training)	0.22	0.36*	0.30	312.7 (18.4%)	348.0 (20.5%)	326.8 (19.2%)
LiDAR	Trees \geq 10cm (biomass not used in training)	0.39*	0.56**	0.48**	241.9 (16.1%)	326.8 (21.7%)	266.0 (17.8%)
LiDAR	Trees \geq 10cm (biomass used in training)	0.51**	0.66**	0.58**	232.0 (15.5%)	349.4 (23.2%)	267.4 (17.8%)

Table 13. Biomass results obtained when all 25 plots were used for testing and training. Two asterisks indicates significance at the 99% level respectively based on a one tailed test.

Imagery	Processing technique	Min. r	Max. r	Avg. r	Min. RMSE (t / ha)	Max. RMSE (t / ha)	Avg. RMSE (t / ha)
LiDAR	Equation derived during optimization	0.71**	0.70**	0.71**	198.1 (15.0%)	200.9 (15.3%)	199.5 (15.1%)
LiDAR	Equation derived after optimization	0.72**	0.71**	0.72**	195.2 (14.8%)	196.6 (15.0%)	196.6 (14.9%)

Residuals for Tree Count Predictions (Photo counts)

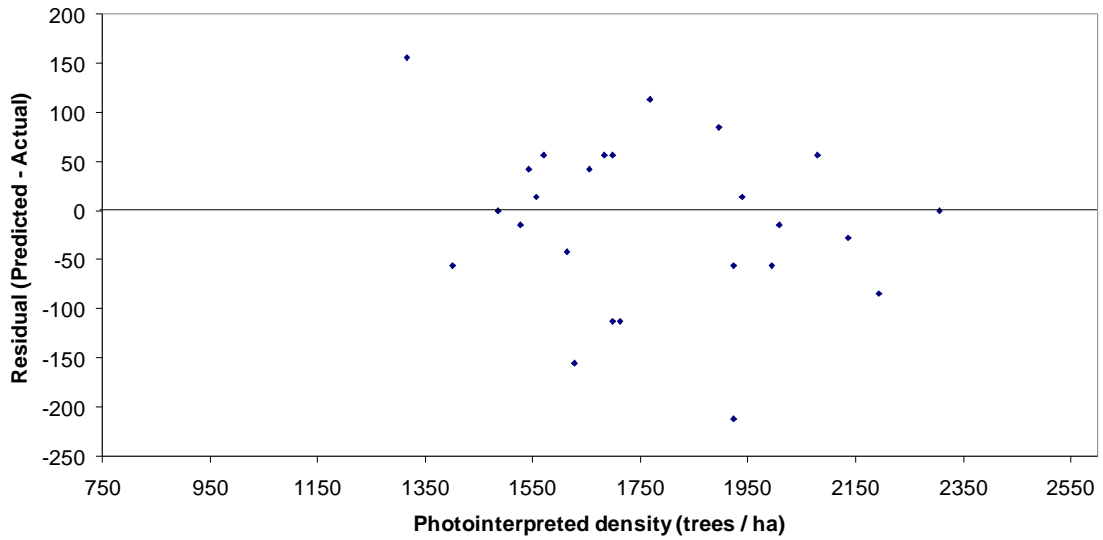


Figure 18a. Residuals when photointerpreted tree counts in all plots were used for training and testing.

Residuals for Tree Count Predictions (DBH > 7cm, Orthophotograph)

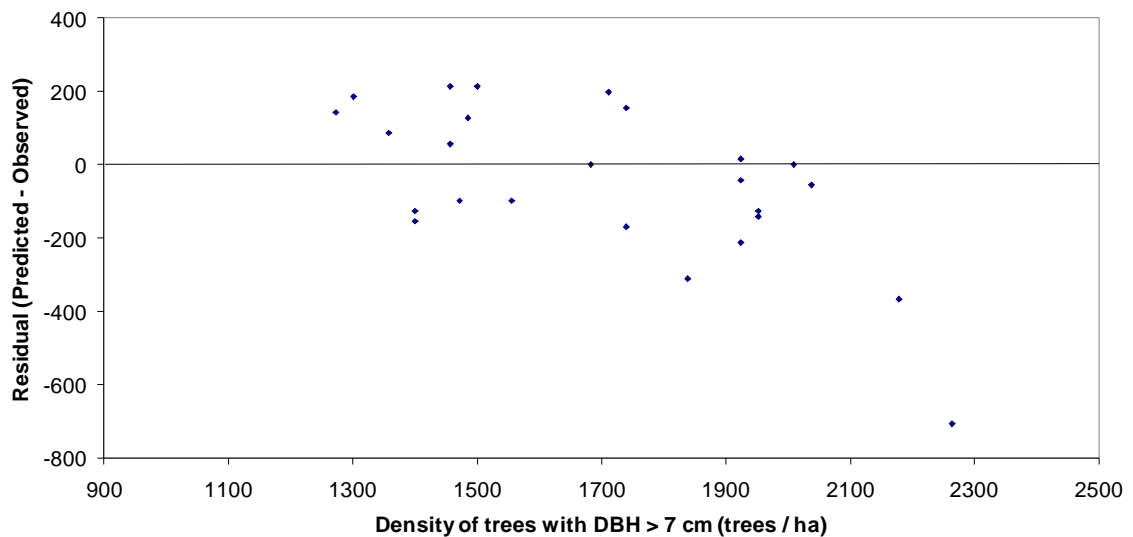


Figure 18b. Residuals when the number of trees with DBH \geq 7 cm in all plots were used for training and testing together with the orthophotograph.

Residuals for Tree Count Predictions (DBH > 10cm, orthophotograph)

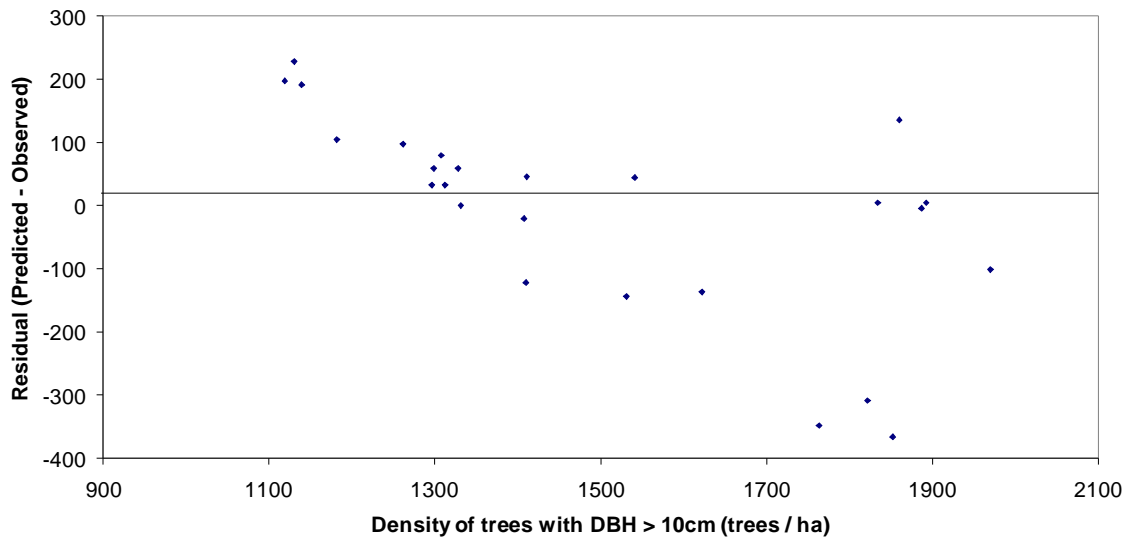


Figure 18c. Residuals when the number of trees with DBH ≥ 10 cm in all plots were used for training and testing together with the orthophotograph.

Residuals for Tree Count Predictions (DBH > 7cm, LiDAR)

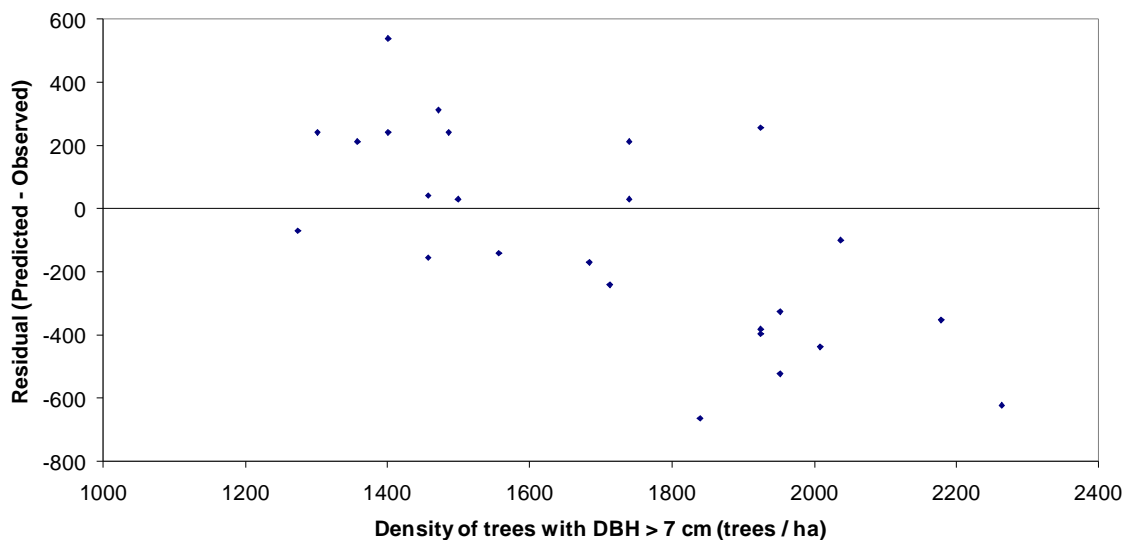


Figure 18d. Residuals when the number of trees with DBH ≥ 7 cm in all plots were used for training and testing together with the LiDAR CHM. Biomass was not simultaneously optimized.

Residuals for Tree Count Predictions (DBH > 10cm, LiDAR)

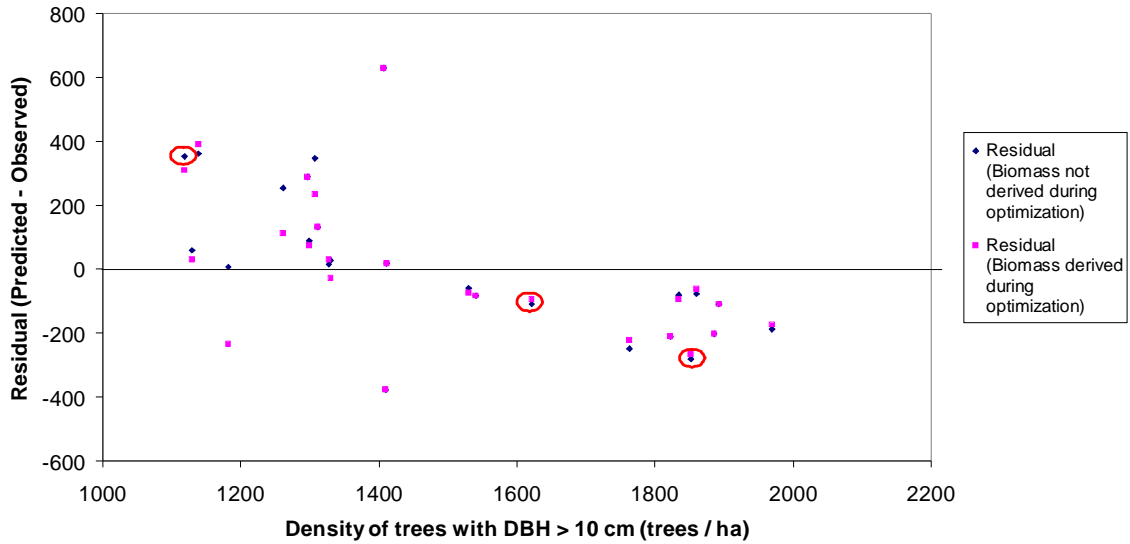


Figure 18e. Residuals when number of trees with DBH ≥ 10 cm in all plots were used for training and testing together with the LiDAR CHM.

Residuals for Biomass Predictions

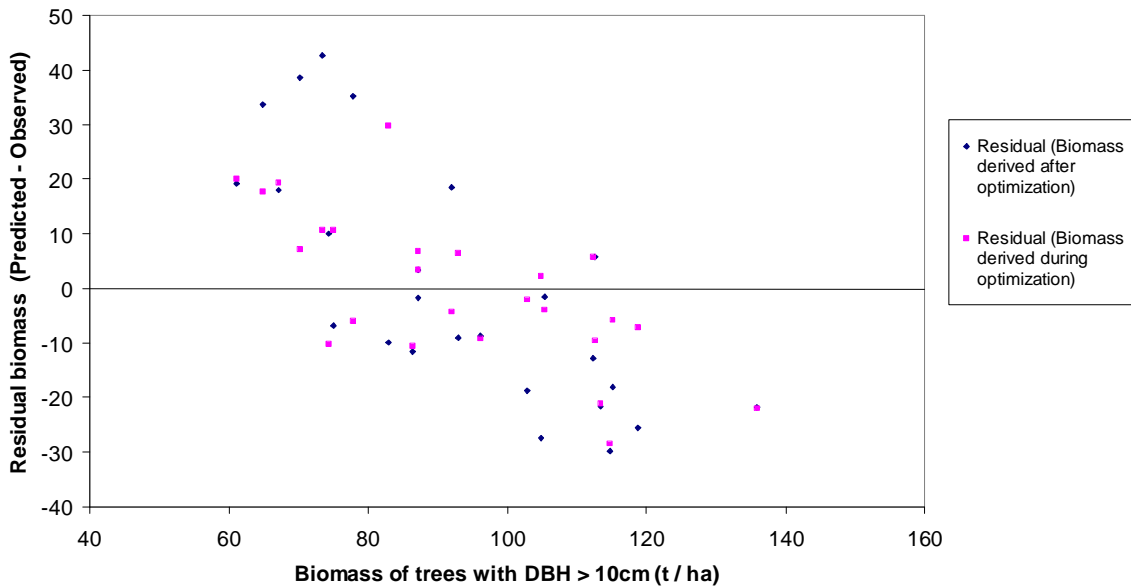


Figure 18f. Residuals when biomass of trees with DBH ≥ 10 cm in all plots were used for training and testing together with the LiDAR CHM. Circled points are shown in Figure 19.

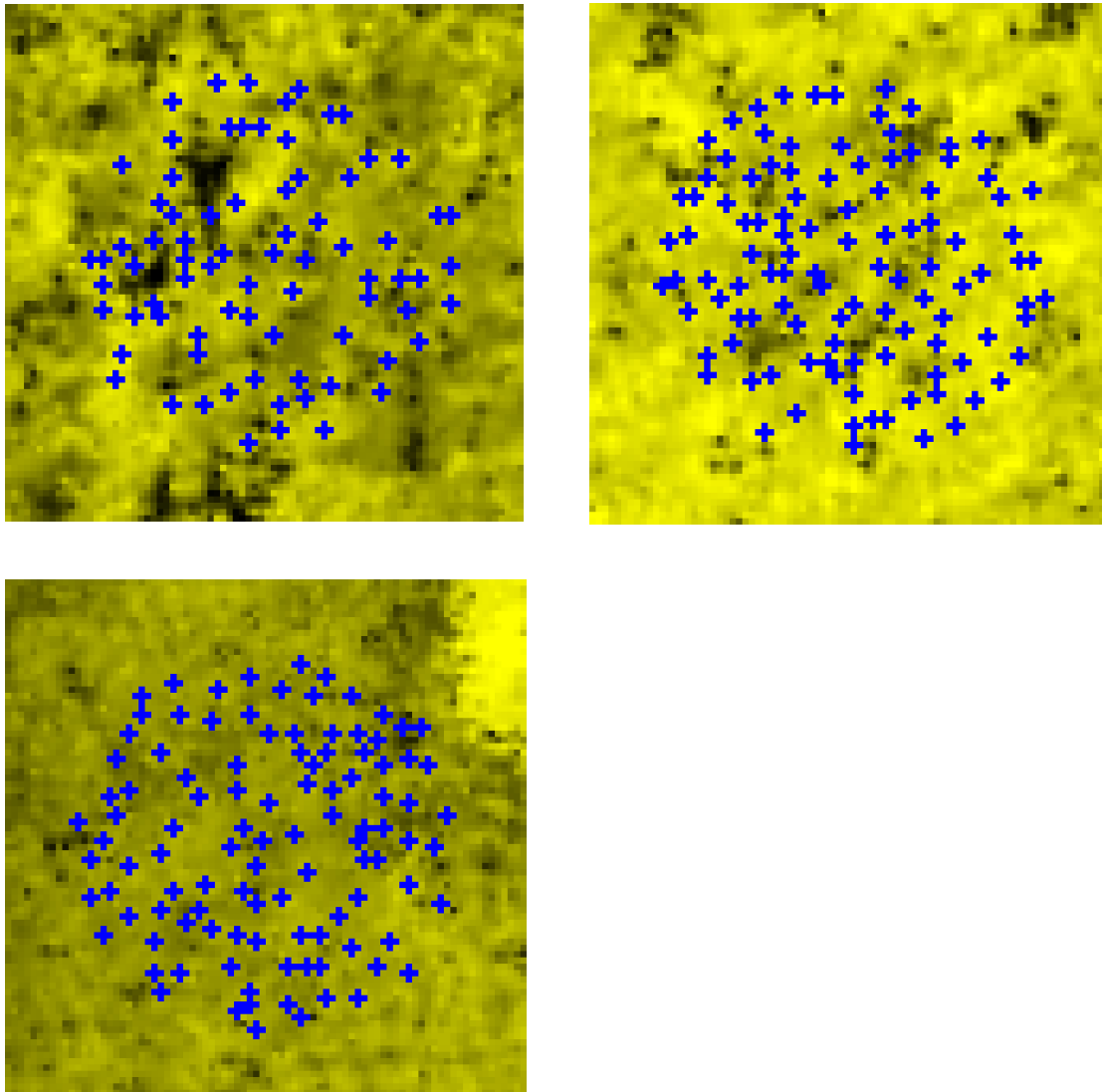


Figure 19. LiDAR images produced after training using all 25 training plots and no biomass data. Blue crosses indicate the locations of trees found by the program. The images correspond to the circles points in Figure 18f, and represent a strong positive residual (upper left, plot 1), a small residual (upper right, plot 2), and a strong negative residual (bottom left, plot 22).

dense understory, leading to higher than average or lower than average density or biomass. However, the human photointerpreter made similar mistakes as shown by the lack of correlation between the residuals and the photointerpreted tree counts (Figure 18a). This problem could be addressed by applying a gain correction based on the training data, hence making the procedure more like a regression analysis. However, the gain correction is only likely to be accurate and improve results if a large number of training images are used or if the non-systematic component of the prediction error is low. It also shifts the approach from being an object-based approach to being statistically-based approach, which may be undesirable. Examining the effects of a gain correction may be an interesting area of future research.

Table 14 shows the Durbin-Watson statistic (Lee *et al.* 2000, p. 712 - 713), which was used to examine first order autocorrelation after regressing the tree counts or biomass values obtained by the program against the training data. This statistic is important, since if autocorrelation existed after regression, the correlation coefficients calculated for these data would be invalid since one of the assumptions of the regression model (uncorrelated errors) would not hold (Lee *et al.*, 2000, p. 711). For 25 samples, values of the Durbin-Watson statistic greater than 1.45 or less than 2.55 indicate that no first order autocorrelation is present (Lee *et al.*, 2000, p. 712). All Durbin-Watson statistics for these data fall within this range, indicating that no first order autocorrelation is present and that the correlation coefficients are valid.

Table 14. Durbin-Watson statistics for the runs that were trained and tested using all plots.

Imagery	Variable	Trees	Durbin-Watson statistic
Orthophotograph	Tree count	Photointerpreted	1.865
Orthophotograph	Tree count	DBH \geq 7cm	1.884
Orthophotograph	Tree count	DBH \geq 10cm	1.935
LiDAR (biomass equation not derived during optimization)	Tree count	DBH \geq 7cm	1.598
LiDAR (biomass equation not derived during optimization)	Tree count	DBH \geq 10cm	1.799
LiDAR (biomass equation derived during optimization)	Tree count	DBH \geq 10cm	1.987
LiDAR (biomass equation not derived during optimization)	Biomass	DBH \geq 10cm	1.460
LiDAR (biomass equation derived during optimization)	Biomass	DBH \geq 10cm	2.152

4.3.2 Intercomparison of results obtained using all the plots as training and testing data

4.3.2.1 Intercomparison of tree count results

The tree count results show a number of trends. The first is that in all cases, the results obtained using the aerial photographs are better (higher correlation, lower RMSE) than those obtained using the LiDAR data. This is unexpected considering the current popularity of LiDAR data among remote sensing researchers. However, it bodes well for practitioners due to the lower cost associated with aerial photographs in comparison to LiDAR data and the greater availability of existing air photo data. There are two possible explanations for the poor results using the LiDAR data. The first is that point samples are used to obtain the LiDAR canopy height model. This means that many of the highest and lowest points associated with the canopy will be missed, resulting in an overall reduction of relief in the canopy, and errors in predicting the concavity of the surface by the interpolation algorithm (the surface may locally appear to curve too much or too little based upon where it was sampled). It is also likely that smaller trees will be missed on a more-or-less random basis, depending on where the laser pulses hit. In comparison, the orthophotograph represents a complete sample. This means that the concavity of the surface will always be consistent. Although smaller trees may still be missed, their being missed is caused by their size and position in the canopy rather than their not being hit by randomly placed laser pulses. Since trees in aerial photographs are missed due to stand structural factors, if the stands have consistent structures then it may be possible for the algorithm to compensate for the missed trees. It would be harder to do in the LiDAR imagery since trees being missed can be due to structural factors or the arrangement of the laser pulses. This is especially true in the plots used for this study since they are relatively small, limiting the amount of error compensation that can take place.

The next trend is that using the photo data, the program is able to predict the number of trees obtained by a human interpreter very accurately. This prediction accuracy greatly exceeds the prediction accuracies for ground-based tree counts. The high accuracy is because the program uses the same clues a human interpreter uses to identify trees, and is therefore likely to

interpret scenes the same way. The same trees will be also visible to both the human interpreter and computer program. Additional important reason for the higher accuracy are:

1. Even though the GPS data were postprocessed, errors are probably still present in the data because the measurements were recorded under the canopy. This means that there may be multipath effects. Most of the stands were hand planted so the rows are not perfectly parallel, and the within row spacing is not perfectly regular. There are also large canopy gaps in many stands. Together, these mean that even a small error in the GPS measurements could lead to significant errors in tree counts.
2. It was not always possible to extend the tape measure in a straight line from the plot center to the outer edge of the plot, and the tape was sometimes bowed. This was especially problematic in plots with a large number of understory trees. Although an effort was made to minimize these problems, inaccuracy in the ground measurements was probably introduced.
3. The number of trees greater with $DBH \geq 10\text{cm}$ was not directly measured, but was calculated using the number of trees greater with $DBH \geq 7\text{ cm}$ and stem diameter measurements (see Section 3.4). Since stem diameter measurements were not made for all trees in each plot, there is some uncertainty associated with the estimated number of trees greater with $DBH \geq 10\text{ cm}$.
4. There is a four or five year temporal discrepancy between the orthophotograph and ground collection dates. This is not considered to be an important consideration due to the very low incidence of snags within the stands that were measured.

For both the orthophoto and the LiDAR data, accuracies were higher for tree counts in which the number of trees with $DBH \geq 10\text{cm}$ was predicted than when the number of trees with $DBH \geq 7\text{cm}$. was predicted. This was expected since 1) crown width and DBH are correlated, so trees with smaller DBH may have crowns that are too small to be identified by the program, and 2) the smaller trees were often occluded by larger trees. However, in the case of the orthophotos, the difference in prediction accuracy seems to be entirely due to an outlier (standardized residual

**Actual vs. Predicted Number of Trees Per Stand, DBH \geq 7cm
Trained Using Data from All Plots**

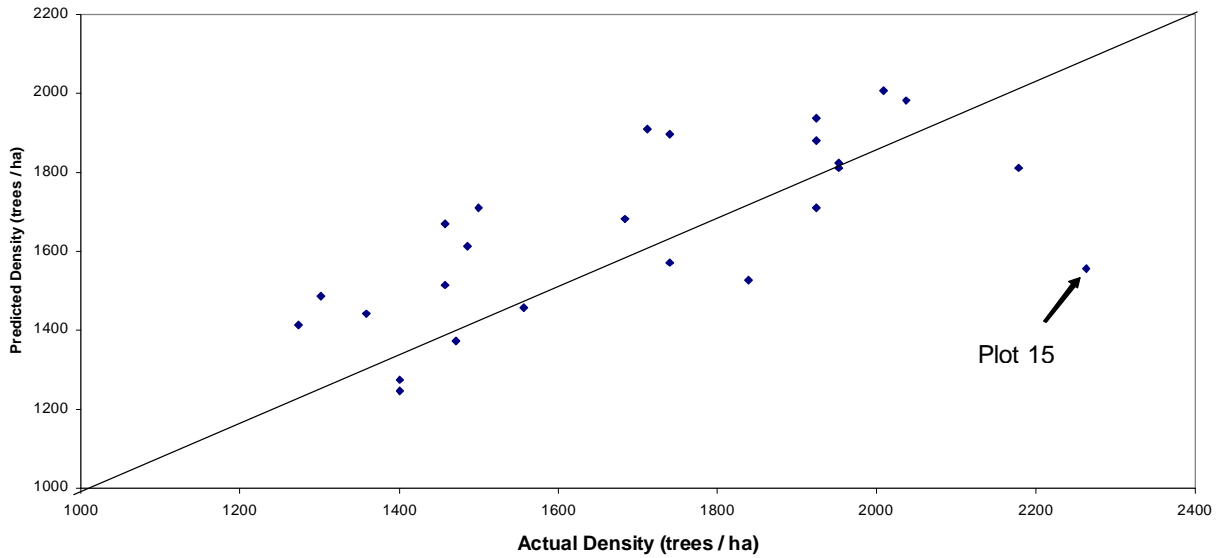


Figure 20: A plot showing the actual versus the predicted number of trees for the most accurate solution set obtained when training was conducted using ground based tree counts of all 25 plots (DBH \geq 7 cm) and the corresponding air photo imagery. The line illustrates a 1:1 relationship. From this plot, it is clear that Plot 15 is an outlier. This is supported by the standardized residual for this point, which is greater than three.

> 3) in all five solution sets, plot 15 (Figure 20). This plot contained a larger fraction of trees with $7 \text{ cm} \leq \text{DBH} \leq 10 \text{ cm}$ than the other plots. When this plot is removed from the data from the most successful run in which the number of trees with $\text{DBH} \geq 7 \text{ cm}$ were used as training, an r of 0.79 and an RMSE of 11.8 trees (10.0%) was found. These values are comparable to the values obtained when the number of trees with $\text{DBH} \geq 10 \text{ cm}$ was used for training and testing. It was not possible to identify this plot as a clear outlier in the LiDAR results in which the number of trees with $\text{DBH} \geq 7 \text{ cm}$ was used for training and testing because the correlation between the actual and predicted values is very low.

4.3.2.2 Intercomparison of biomass results

The biomass results show that the biomass could be predicted with moderate accuracy. Interestingly, the two biomass prediction methods produced nearly identical results, although the method in which the relationship between LiDAR derived values and biomass was developed after training performed slightly better. This was somewhat surprising considering that the methods were very different from one another. Figure 21 shows the results from runs using the two biomass prediction methods, and illustrates the similarity of the results. There was very little within-group variation in r and RMSE values for both methods.

4.3.3 Comparison to human photointerpreter

4.3.3.1 Determining whether the same trees were found

Figures 22a to 22y show results obtained by the algorithm after being trained using human-derived tree counts (from all 25 plots) superimposed on an image showing the locations of trees found by the human interpreter. The accuracy of the tree locations was determined by randomly selecting 100 trees found by the computer (four per image) and seeing whether they corresponded to trees found by the human interpreter, and by finding 100 trees found by the human interpreter and seeing whether they correspond to trees found by the computer. The results showed that 89% of the trees found by the computer corresponded to trees found by the human photointerpreter, and 74% of the trees found by human interpreter corresponded to trees (text continues on page 116)

**Actual vs. Predicted Biomass, DBH = 10cm
Trained Using Data from All Plots**

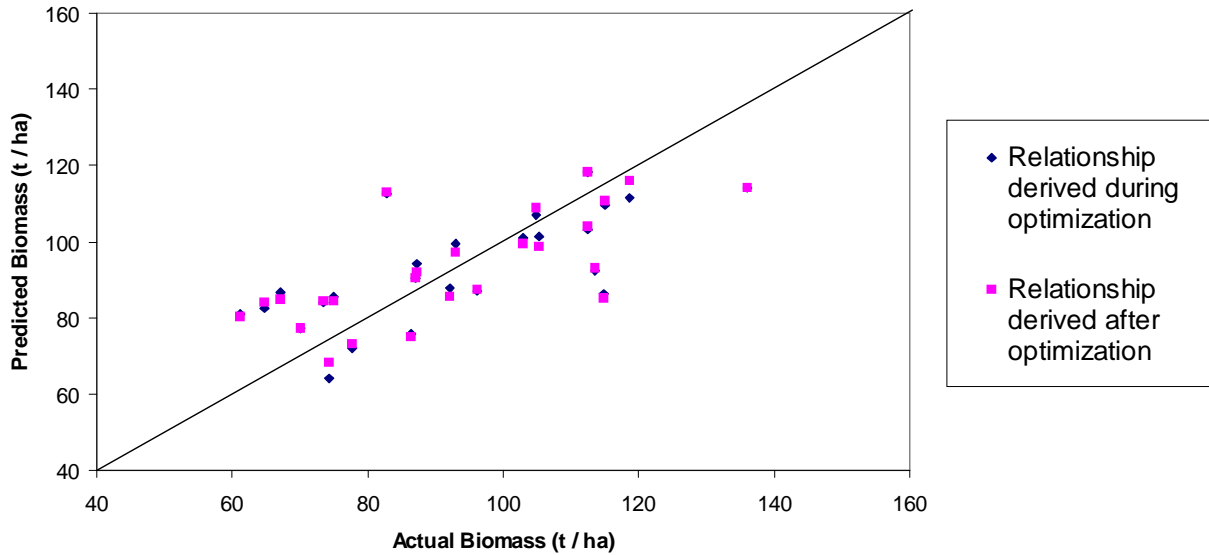


Figure 21. A plot showing the actual versus the predicted biomass trees for the most accurate solution set obtained when training was conducted using ground biomass estimates of all 25 plots (DBH \geq 10 cm) and the corresponding LiDAR imagery. Results using both biomass prediction methods described in the text are shown. The line illustrates a 1:1 relationship.

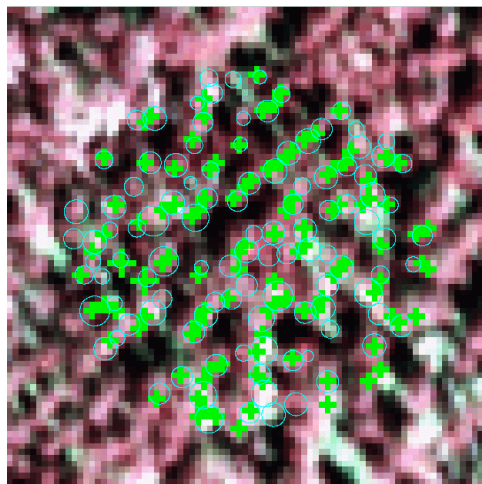
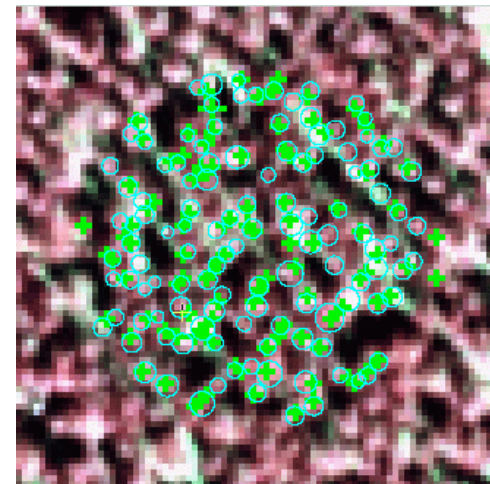
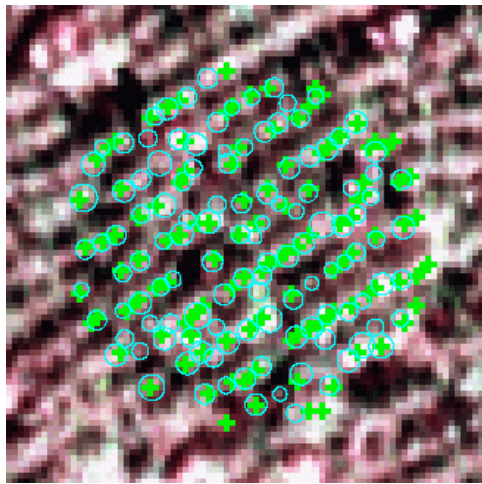
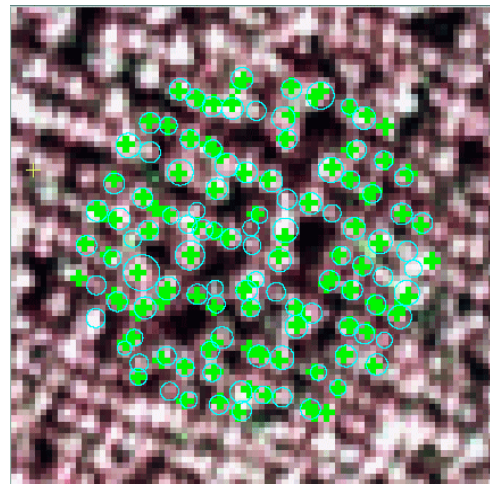
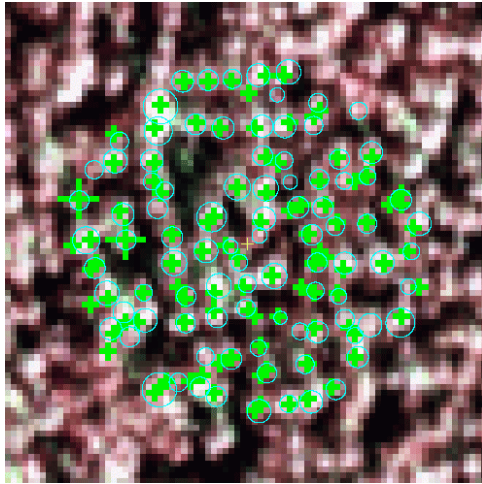


Figure 22a - e. Aerial photograph segments corresponding to plots 1 (upper left), 2 (upper center), 3 (upper right), 4 (middle left), and 5 (lower left). Trees found by the computer after training using human tree counts from all 25 plots are shown as green crosses. The cyan circles denote tree boundaries found by the human interpreter.

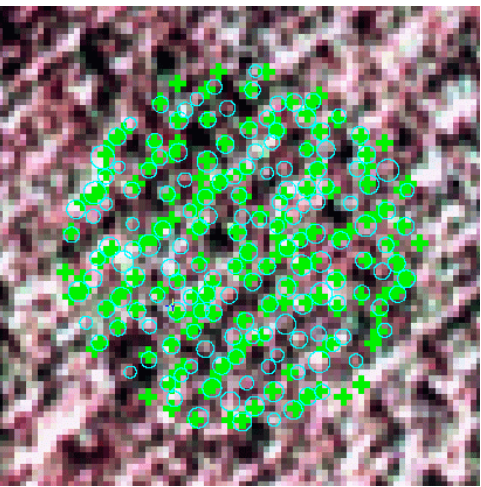
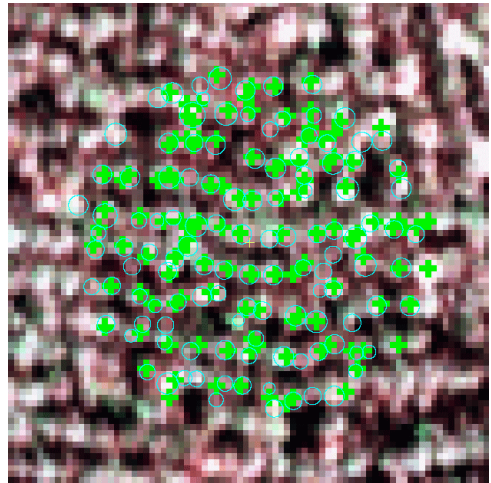
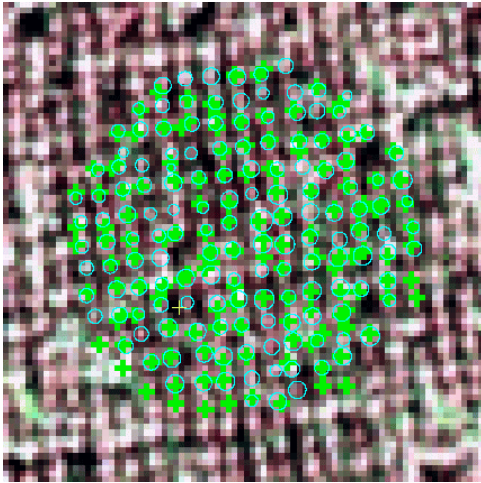
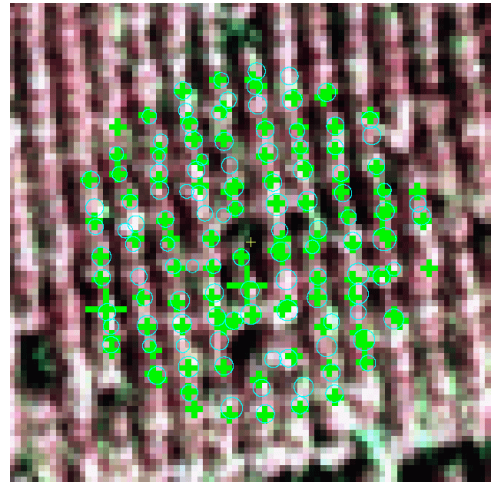
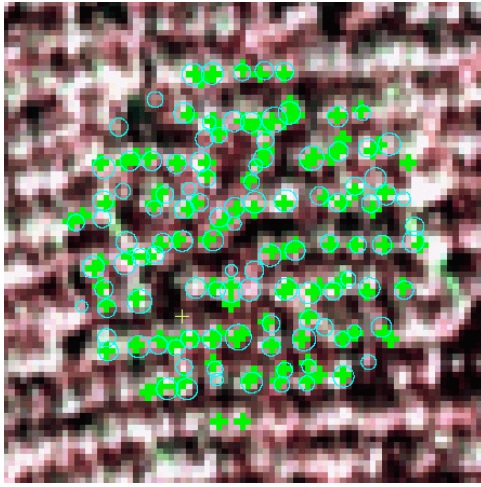


Figure 22f - j. Aerial photograph segments corresponding to plots 6 (upper left), 7 (upper center), 8 (upper right), 9 (middle left), and 10 (lower left). Trees found by the computer after training using human tree counts from all 25 plots are shown as green crosses. The cyan circles denote tree boundaries found by the human interpreter.

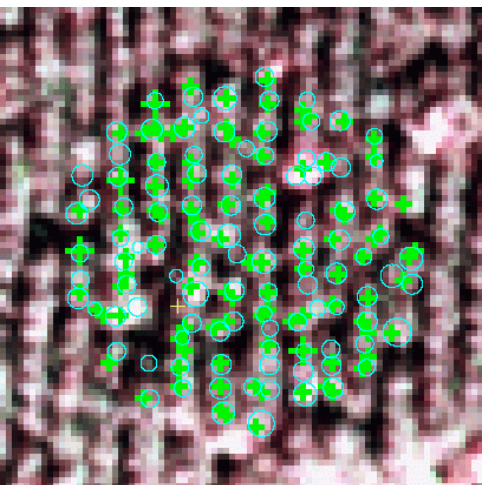
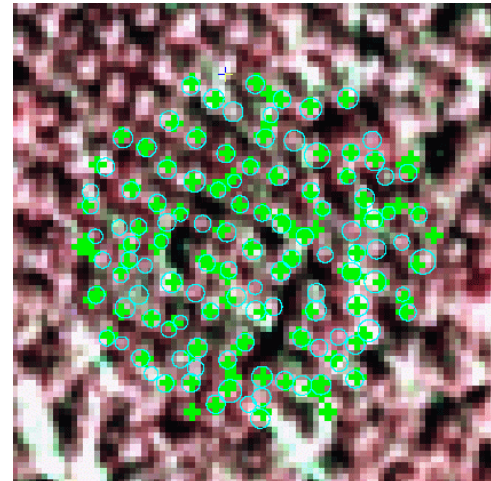
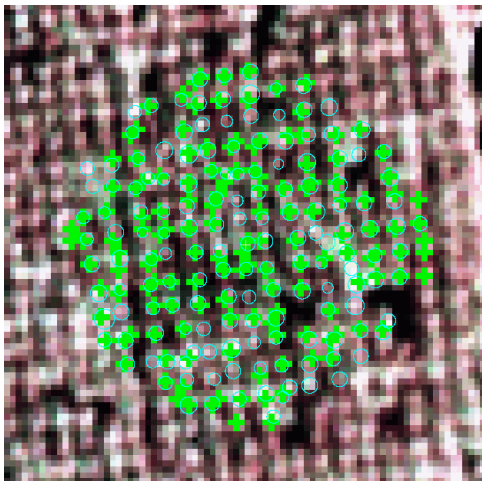
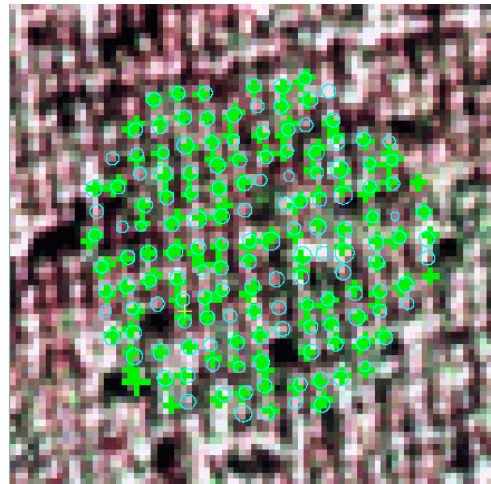
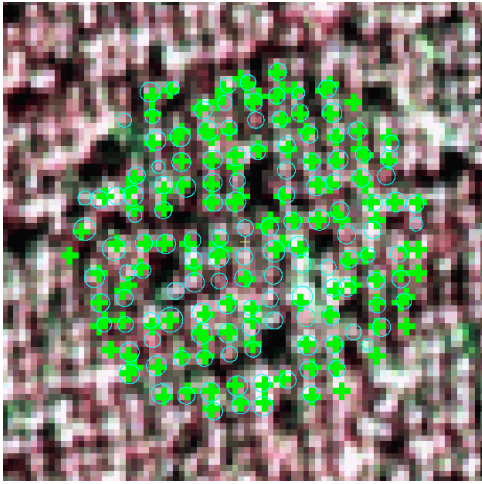


Figure 22k - o. Aerial photograph segments corresponding to plots 11 (upper left), 12 (upper center), 13 (upper right), 14 (middle left), and 15 (lower left). Trees found by the computer after training using human tree counts from all 25 plots are shown as green crosses. The cyan circles denote tree boundaries found by the human interpreter.

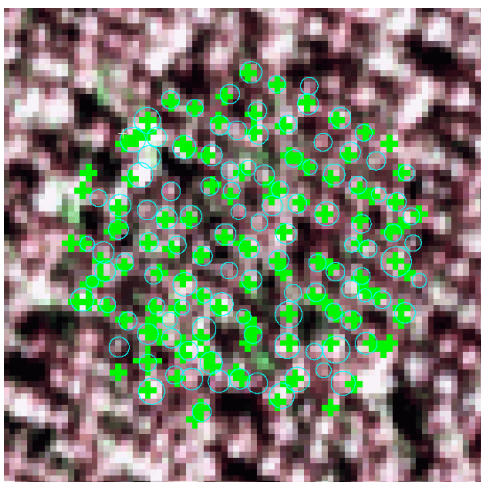
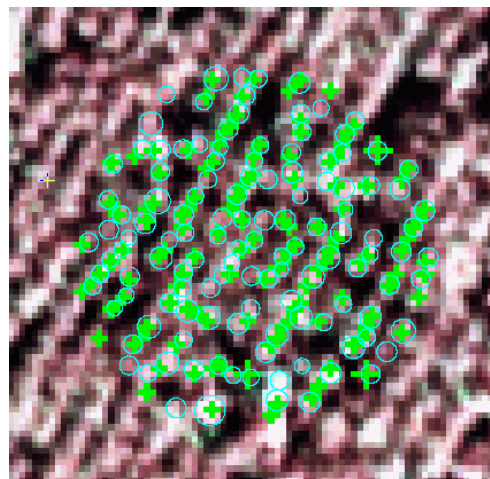
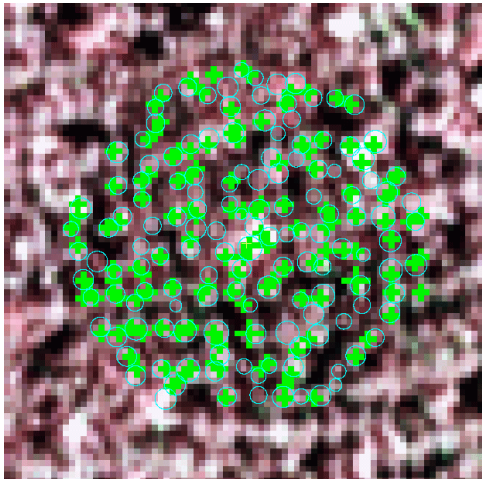
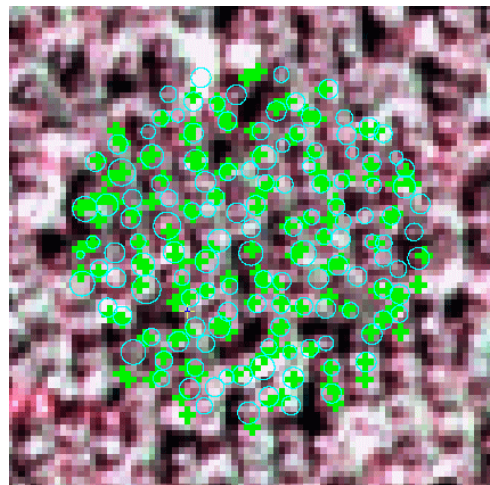
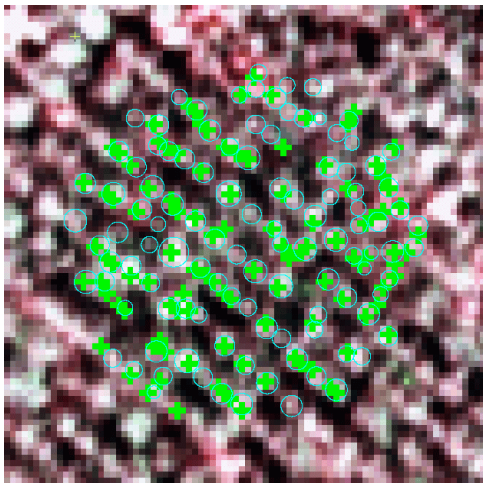


Figure 22p - t. Aerial photograph segments corresponding to plots 16 (upper left), 17 (upper center), 18 (upper right), 19 (middle left), and 20 (lower left). Trees found by the computer after training using human tree counts from all 25 plots are shown as green crosses. The cyan circles denote tree boundaries found by the human interpreter.

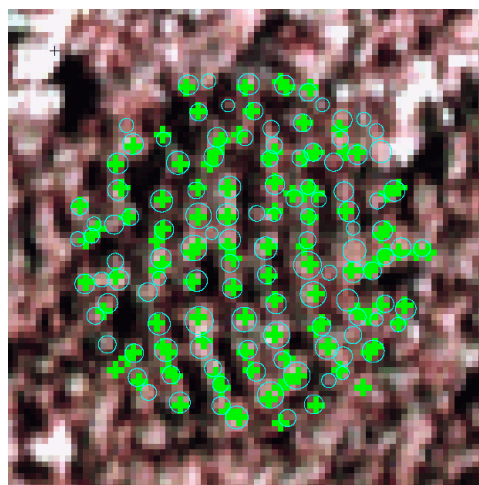
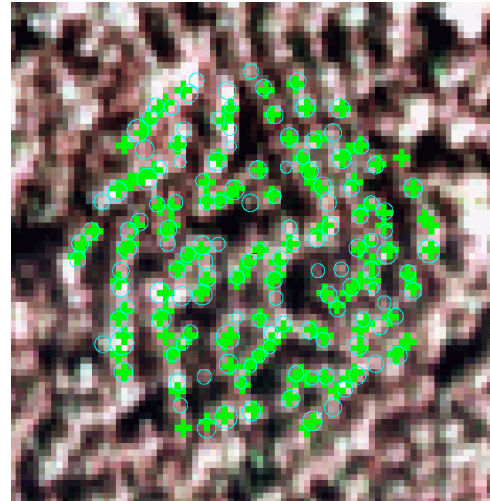
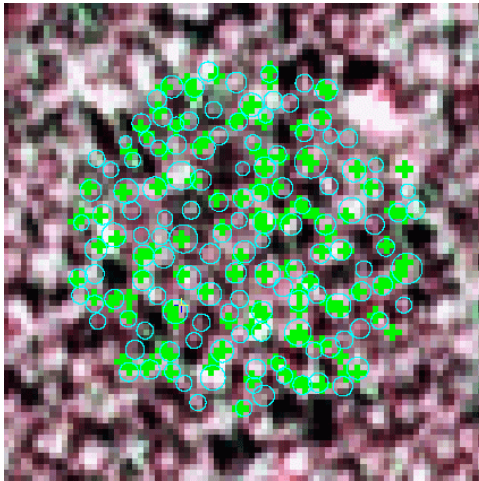
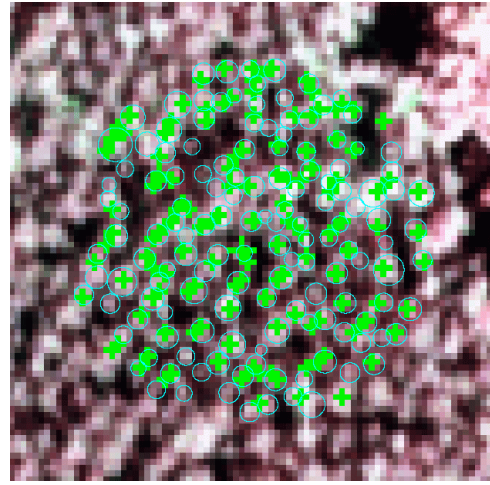
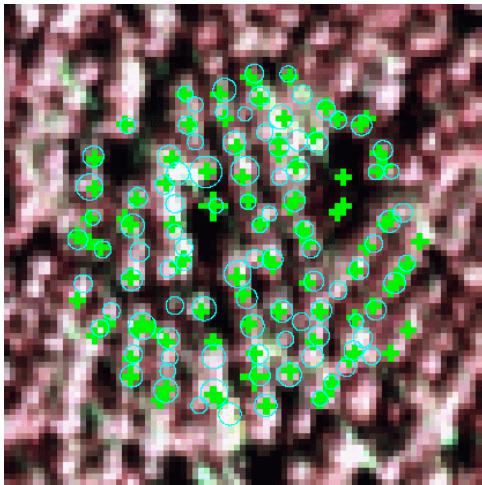


Figure 22u - y. Aerial photograph segments corresponding to plots 21 (upper left), 22 (upper center), 23 (upper right), 24 (middle left), and 25 (lower left). Trees found by the computer after training using human tree counts from all 25 plots are shown as green crosses. The cyan circles denote tree boundaries found by the human interpreter.

found by the computer. This indicates that for the most part, the computer vision algorithm was successful in replicating the way in which humans look at aerial images of pine plantations. The reason why the percentage of trees found by the human interpreter that correspond to trees found by the computer is lower than the percentage of trees found by the computer that correspond to trees found by the human interpreter is that the program has a tendency to find two or more trees in single trees found by the human interpreter, and to miss small trees found by the human interpreter. This problem may decrease if more training data are used, since there would be fewer ways for the program to achieve a low (good) fitness score while still incorrectly identifying trees.

4.3.3.2 Comparison of ground count predictions

The prediction accuracies for ground counts obtained by the human photointerpreter are shown in Table 15. By comparing this table and the photo-based results from Table 12, it can be seen that after training, the computer vision algorithm is capable of producing more accurate tree counts than a human interpreter who had no site-specific training. Although it is likely that the human interpreter's results would improve with site specific training, this would take time, and in practice would cost more money than using a computer to perform the analysis.

Table 15. Accuracies of tree counts obtained by a human interpreter, based on ground counts from all 25 plots. One and two asterisks indicate significance at the 95%, and 99% level respectively based on a one tailed test.

Statistic	DBH \geq 7 cm	DBH \geq 10 cm
r	0.46*	0.67**
RMSE (trees / ha)	292.8 (17.2%)	352.3 (23.5%)

4.4 Analysis of the computer vision algorithm after training and testing using subsets of the data

The results of training using subsets of the plot data for training and testing are shown in Tables 16, 17, and 18. These results are valuable because they are representative of the results a practitioner would expect to obtain, since he or she would not know the density or biomass of the testing plots, and therefore could not use them as training data. Interestingly, these tables show that the prediction accuracies can be both higher and lower than the accuracies obtained when all twenty five plots were used for training and testing, which is probably due to fortuitous or unfortuitous choices of training and testing subsamples. Although in many cases patterns are hard to discern in these data, several trends do appear.

4.4.1 Effects of training sample size on accuracy

4.4.1.1 Effect on tree count accuracies

Although there is a wide range in correlation coefficients and RMSE values for the tree count predictions, there is no clear connection between the number of training samples and the resulting accuracy. This is somewhat surprising, but is likely due to the relatively small samples used and the inclusion of outliers in some of the training and testing datasets. If a larger dataset were used, it is likely that using more training data would result in more accurate predictions since outlying points would have less of an effect.

4.4.1.2 Effect on biomass prediction accuracies

Looking at the data, there is no consistent relationship between the correlation coefficient and the number of training samples. However, in all cases the average RMSE for the predicted biomass is higher when three training samples are used during optimization than when six or twelve are used. This makes sense, because in both cases the randomly selected samples for three sample runs cover only a small portion of the overall biomass range (Figure 23), resulting in the algorithm having to make predictions of biomass values that are well outside the range of the training data. The samples for six and twelve sample runs are much more representative of the overall range.

Table 16. Tree count results obtained using subsets of the data. In this table CH indicates that the data used for training and testing consisted of counts obtained by a human interpreter. C7 and C10 indicate that ground data consisting of the number of trees with DBH ≥ 7 cm or DBH ≥ 10 cm respectively were used. P indicates a run in which biomass values were determined after optimization, and D indicates a run in which biomass values were determined during optimization. One or two asterisks indicate significance at the 95% or 99% level respectively based on a one tailed test.

Imagery	Predicted data	Training set	Number of training images	Low r	High r	Average r	Low RMSE (trees / ha)	High RMSE (trees / ha)	Average RMSE (trees / ha)
Photo	CH	1	3	0.78**	0.89**	0.84**	107.5 (6.1%)	166.9 (9.5%)	135.8 (7.8%)
Photo	CH	1	6	0.70**	0.90**	0.81**	107.5 (6.1%)	210.8 (12.0%)	134.4 (7.6%)
Photo	CH	1	12	0.79**	0.85**	0.82**	127.3 (7.2%)	147.1 (8.4%)	138.6 (7.9%)
Photo	CH	2	3	0.62*	0.82**	0.76**	150.0 (8.6%)	451.3 (25.9%)	246.2 (14.2%)
Photo	CH	2	6	0.66**	0.80**	0.75**	145.7 (8.4%)	241.9 (13.9%)	175.4 (10.1%)
Photo	CH	2	12	0.68**	0.87**	0.79**	161.3 (9.2%)	199.5 (11.5%)	183.9 (10.5%)
Photo	C7	1	3	0.60*	0.78**	0.71**	171.2 (10.5%)	251.8 (15.5%)	216.5 (13.3%)
Photo	C7	1	6	0.29	0.68**	0.53*	240.5 (14.7%)	340.9 (20.9%)	280.1 (17.2%)
Photo	C7	1	12	0.31	0.70**	0.56*	266.0 (16.3%)	328.2 (20.2%)	288.6 (17.7%)
Photo	C7	2	3	0.06	0.41	0.24	280.1 (16.3%)	370.7 (21.5%)	329.6 (19.1%)
Photo	C7	2	6	0.13	0.49*	0.27	210.8 (12.2%)	585.7 (33.9%)	353.7 (20.5%)
Photo	C7	2	12	0.04	0.34	0.21	164.1 (9.5%)	374.9 (21.7%)	326.8 (18.9%)
Photo	C10	1	3	0.32	0.90**	0.62*	159.9 (11.2%)	502.2 (35.2%)	316.9 (22.2%)
Photo	C10	1	6	0.45	0.76**	0.66**	263.1 (18.4%)	339.5 (23.7%)	288.6 (20.3%)
Photo	C10	1	12	0.40	0.78**	0.63*	226.4 (15.8%)	314.1 (22.0%)	277.3 (19.4%)
Photo	C10	2	3	0.35	0.61*	0.51*	223.5 (14.8%)	391.9 (26.0%)	297.1 (19.7%)
Photo	C10	2	6	0.47	0.57*	0.51*	213.6 (14.2%)	377.7 (25.1%)	240.5 (15.9%)
Photo	C10	2	12	0.47	0.60*	0.52*	247.6 (15.2%)	325.4 (21.6%)	278.7 (18.5%)

LiDAR	C7 (P)	1	3	0.32	0.63*	0.45	333.9 (20.5%)	35.9 (31.2%)	507.9 (24.6%)
LiDAR	C7 (P)	1	6	0.55*	0.78**	0.65**	311.2 (19.1%)	37.3 (32.4%)	527.7 (23.2%)
LiDAR	C7 (P)	1	12	0.65**	0.78**	0.73**	222.1 (13.6%)	31.5 (27.3%)	445.6 (20.0%)
LiDAR	C7 (P)	2	3	-0.50	-0.09	-0.26	430.1 (24.9%)	41.8 (34.3%)	591.3 (29.1%)
LiDAR	C7 (P)	2	6	-0.51	0.31	-0.06	475.3 (27.5%)	50.4 (41.3%)	713.0 (34.2%)
LiDAR	C7 (P)	2	12	-0.17	0.45	0.00	360.8 (20.9%)	38.3 (31.4%)	541.8 (25.8%)
LiDAR	C10 (P)	1	3	0.25	0.45	0.37	328.2 (23.0%)	26.3 (26.0%)	372.1 (24.6%)
LiDAR	C10 (P)	1	6	0.21	0.59*	0.42	241.9 (17.0%)	74.4 (73.7%)	1052.5 (48.1%)
LiDAR	C10 (P)	1	12	0.31	0.65**	0.47	222.1 (15.5%)	23.7 (23.5%)	335.3 (19.2%)
LiDAR	C10 (P)	2	3	-0.07	0.48*	0.36	367.8 (24.4%)	77.1 (72.3%)	1090.7 (49.5%)
LiDAR	C10 (P)	2	6	-0.11	0.48*	0.23	297.1 (19.7%)	44.8 (42.0%)	633.8 (29.1%)
LiDAR	C10 (P)	2	12	-0.17	0.36	0.11	319.7 (21.2%)	35.7 (26.2%)	505.1 (33.5%)
LiDAR	C10 (D)	1	3	0.28	0.61*	0.44	318.3 (22.3%)	25.2 (25.0%)	356.5 (24.1%)
LiDAR	C10 (D)	1	6	0.30	0.63*	0.42	274.5 (19.2%)	30.7 (30.4%)	434.4 (23.8%)
LiDAR	C10 (D)	1	12	0.55*	0.66**	0.61*	222.1 (15.6%)	21.8 (21.6%)	308.4 (17.6%)
LiDAR	C10 (D)	2	3	-0.15	0.55*	0.16	338.1 (22.4%)	60.9 (57.1%)	861.6 (30.8%)
LiDAR	C10 (D)	2	6	0.44	0.50*	0.48*	602.7 (40.0%)	45.9 (43.1%)	649.4 (41.7%)
LiDAR	C10 (D)	2	12	-0.15	0.25	0.08	324.0 (21.4%)	36.4 (34.2%)	515.0 (25.5%)

Table 17. Individual plot tree density estimates obtained using six training images (training set 1). The values in this chart represent the average of the five density estimates obtained using the five sets of parameters produced by the Nelder-Mead simplex algorithm. Errors are in terms of predicted density - actual density. P indicates a run in which biomass values were determined after optimization, and D indicates a run in which biomass values were determined during optimization. All density values are in trees per hectare.

Orthophotograph (Photointerpreted density)			
Plot	Estimated density	Error	Error (%)
1	1601.5	285.8	21.7
2	1593.0	65.1	4.3
3	1774.0	76.4	4.5
4	1544.9	-166.9	-9.8
5	1567.5	25.5	1.7
6	1626.9	56.6	3.6
7	1576.0	-50.9	-3.1
8	2113.6	-22.6	-1.1
9	1779.7	96.2	5.7
10	2325.8	19.8	0.9
11	1980.6	42.4	2.2
12	2164.5	84.9	4.1
13	2139.0	-53.8	-2.5
14	1559.0	-53.8	-3.3
15	1570.3	84.9	5.7
16	1561.8	5.7	0.4
17	2110.7	215.0	11.3
18	2096.6	87.7	4.4
19	1873.1	104.7	5.9
20	1813.7	158.4	9.6
21	1397.7	-2.8	-0.2
22	1960.8	-34.0	-1.7
23	1779.7	-144.3	-7.5
24	1858.9	-65.1	-3.4
25	1556.2	-141.5	-8.3

Orthophotograph (DBH > 7 cm)			
Plot	Estimated density	Error	Error (%)
1	1607.1	150.0	10.3
2	1694.8	393.3	30.2
3	1808.0	124.5	7.4
4	1525.1	166.9	12.3
5	1502.4	229.2	18.0
6	1629.7	-110.3	-6.3
7	1474.1	2.8	0.2
8	1974.9	-62.2	-3.1
9	1759.9	260.3	17.4
10	2048.5	39.6	2.0
11	1887.2	-65.1	-3.3
12	1983.4	59.4	3.1
13	1952.3	28.3	1.5
14	1508.1	107.5	7.7
15	1646.7	-616.8	-27.3
16	1686.3	-152.8	-8.3
17	1907.0	195.2	11.4
18	2006.1	266.0	15.3
19	1731.6	-192.4	-10.0
20	1884.4	-67.9	-3.5
21	1411.9	11.3	0.8
22	1912.7	455.5	31.3
23	1731.6	246.2	16.6
24	1759.9	-418.8	-19.2
25	1544.9	-11.3	-0.7

Orthophotograph (DBH > 10cm)			
Plot	Estimated density	Error	Error (%)
1	1417.5	287.5	25.4
2	1542.0	403.2	35.4
3	1737.3	206.8	13.5
4	1414.7	153.6	12.2
5	1335.5	153.2	13.0
6	1519.4	220.5	17.0
7	1386.4	-20.9	-1.5
8	1915.5	29.2	1.6
9	1649.6	530.5	47.4
10	1963.6	-5.9	-0.3
11	1873.1	39.1	2.1
12	1960.8	68.8	3.6
13	1884.4	24.5	1.3
14	1420.4	113.2	8.7
15	1406.2	-216.0	-13.3
16	1406.2	-3.8	-0.3
17	1564.7	24.1	1.6
18	1813.7	486.5	36.7
19	1604.3	-159.4	-9.0
20	1669.4	-152.8	-8.4
21	1304.4	-26.2	-2.0
22	1649.6	338.1	25.8
23	1547.7	136.5	9.7
24	1680.7	-171.2	-9.2
25	1471.3	174.5	13.5

Lidar (D) (DBH > 10 cm)			
Plot	Estimated density	Error	Error (%)
1	1106.3	-23.7	-2.1
2	1332.7	193.8	17.0
3	1530.7	0.3	0.0
4	1584.5	323.4	25.6
5	1145.9	-36.4	-3.1
6	1505.3	206.3	15.9
7	1791.0	383.7	27.3
8	1779.7	-106.6	-5.7
9	1717.5	598.4	53.5
10	1717.5	-252.0	-12.8
11	1839.1	5.1	0.3
12	1553.4	-338.6	-17.9
13	1952.3	92.4	5.0
14	1601.5	294.3	22.5
15	1247.8	-374.4	-23.1
16	1078.0	-332.0	-23.5
17	1324.2	-216.5	-14.0
18	1310.0	-17.2	-1.3
19	1329.8	-433.8	-24.6
20	1338.3	-483.8	-26.6
21	1369.4	38.9	2.9
22	1471.3	159.9	12.2
23	1256.3	-154.9	-11.0
24	1389.2	-462.6	-25.0
25	1440.2	143.4	11.1

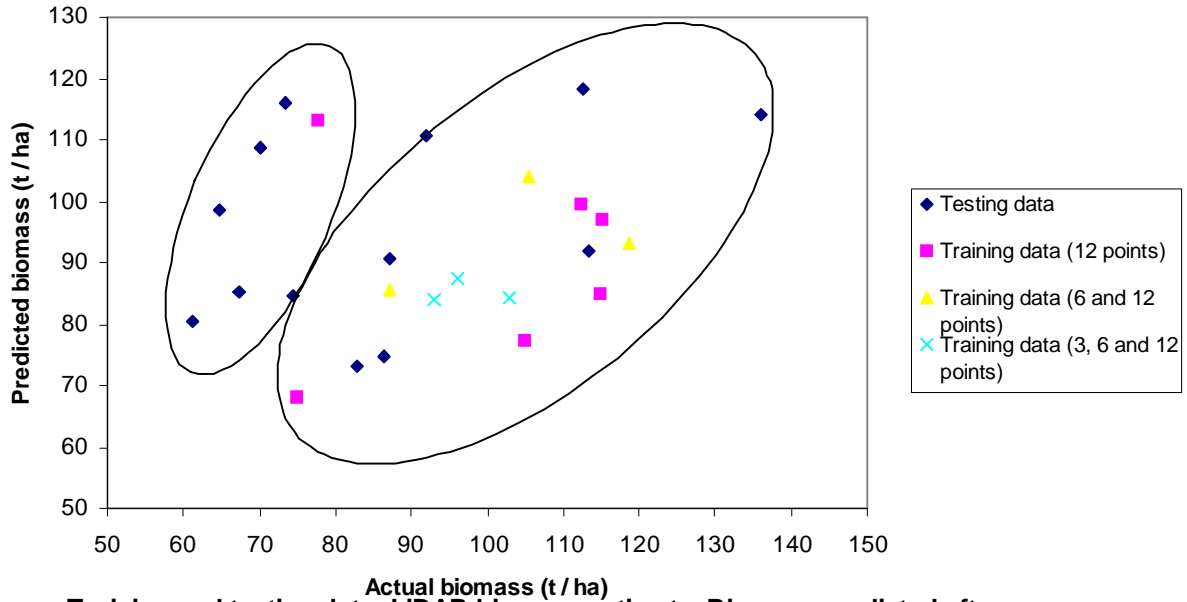
Lidar (P) (DBH > 7 cm)			
Plot	Estimated density	Error	Error (%)
1	1117.6	-339.5	-23.3
2	1454.3	152.8	11.7
3	1508.1	-175.4	-10.4
4	1601.5	243.3	17.9
5	979.0	-294.3	-23.1
6	1714.6	-25.5	-1.5
7	1578.8	107.5	7.3
8	1839.1	-198.1	-9.7
9	1363.8	-135.8	-9.1
10	1451.5	-557.4	-27.7
11	1228.0	-724.3	-37.1
12	1496.8	-427.2	-22.2
13	1887.2	-36.8	-1.9
14	2048.5	647.9	46.3
15	1403.4	-860.1	-38.0
16	1145.9	-693.2	-37.7
17	1230.8	-481.0	-28.1
18	1530.7	-209.4	-12.0
19	1409.1	-515.0	-26.8
20	1377.9	-574.4	-29.4
21	1559.0	158.4	11.3
22	1536.4	79.2	5.4
23	1864.6	379.1	25.5
24	1853.3	-325.4	-14.9
25	1270.4	-285.8	-18.4

Lidar (P) (DBH > 10cm)			
Plot	Estimated density	Error	Error (%)
1	594.2	-535.9	-47.4
2	1561.8	423.0	37.1
3	1791.0	260.6	17.0
4	390.5	-870.7	-69.0
5	220.7	-961.6	-81.3
6	831.9	-467.1	-36.0
7	1556.2	148.9	10.6
8	1833.5	-52.8	-2.8
9	772.4	-346.7	-31.0
10	1881.6	-87.9	-4.5
11	1086.5	-747.5	-40.8
12	1225.1	-666.8	-35.2
13	2068.3	208.4	11.2
14	523.4	-783.7	-60.0
15	1457.2	-165.0	-10.2
16	1131.8	-278.2	-19.7
17	1485.4	-55.2	-3.6
18	585.7	-741.5	-55.9
19	812.0	-951.6	-54.0
20	1539.2	-282.9	-15.5
21	874.3	-456.2	-34.3
22	1751.4	440.0	33.5
23	831.9	-579.3	-41.1
24	1205.3	-646.5	-34.9
25	800.7	-496.1	-38.3

Table 18. Biomass results obtained from the LiDAR imagery using subsets of the data for training and testing. P indicates a run in which biomass values were determined after optimization, and D indicates a run in which biomass values were determined during optimization. One or two asterisks indicate significance at the 95% or 99% level respectively based on a one tailed test.

Processing method	Training set	Number of training images	Low r	High r	Average r	Low RMSE (t/ha)	High RMSE (t/ha)	Average RMSE (t/ha)
P	1	3	0.65**	0.67**	0.66**	22.1 (25.6%)	22.5 (26.1%)	22.3 (25.9%)
P	1	6	0.59**	0.79**	0.64**	17.3 (20.1%)	21.6 (23.8%)	18.2 (21.1%)
P	1	12	0.66**	0.68**	0.67**	18.4 (21.3%)	18.6 (21.6%)	18.5 (21.5%)
P	2	3	0.79**	0.80**	0.80**	26.4 (26.8%)	120.0 (121.7%)	56.8 (57.6%)
P	2	6	0.78**	0.80**	0.79**	21.6 (21.9%)	24.3 (28.4%)	23.3 (23.6%)
P	2	12	0.78**	0.79**	0.78**	15.6 (16.1%)	16.1 (15.8%)	15.8 (16.3%)
D	1	3	0.49*	0.65**	0.57*	17.2 (19.9%)	81.9 (95.0%)	59.9 (69.5%)
D	1	6	0.60*	0.63*	0.62**	17.7 (21.9%)	19.8 (23.0%)	18.9 (21.9%)
D	1	12	0.61*	0.65**	0.63*	19.1 (22.2%)	20.0 (23.1%)	19.5 (22.7%)
D	2	3	0.68**	0.78**	0.75**	43.7 (44.9%)	71.0 (79.6%)	56.0 (56.8%)
D	2	6	0.68**	0.70**	0.69**	27.0 (27.3%)	28.1 (28.5%)	27.6 (28.0%)
D	2	12	0.15	0.54*	0.41	19.3 (19.6%)	22.9 (23.2%)	20.6 (20.9%)

Training and testing data: LiDAR biomass estimate, Biomass predicted after optimization, Set 1



Training and testing data: LiDAR biomass estimate, Biomass predicted after optimization, Set 2

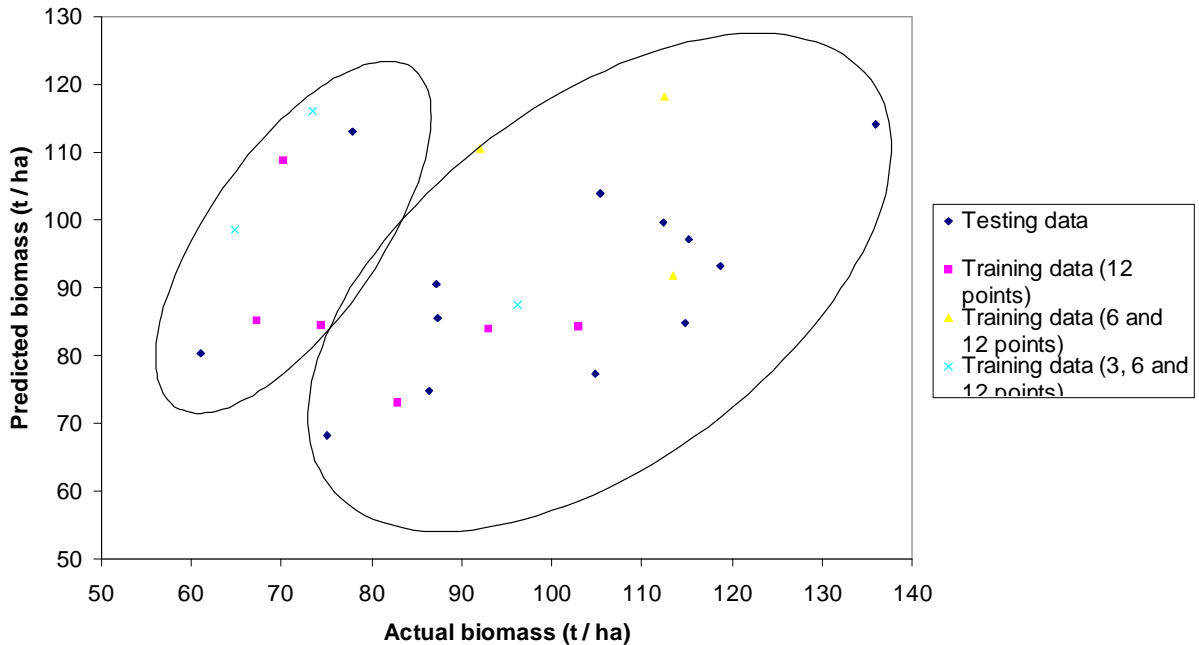


Figure 23. Plots constructed using the biomass of trees with $DBH \geq 10$ cm, and predictions of biomass made when the algorithm was optimized using all 25 plots and the biomass equation was derived after optimization. The upper plot show the points in training set 1, and the lower plot shows the points in training set 2. The ovals indicate the boundaries of elongate clusters seen in the data (see Section 4.4.2.2).

4.4.2 Comparison of the two training sets

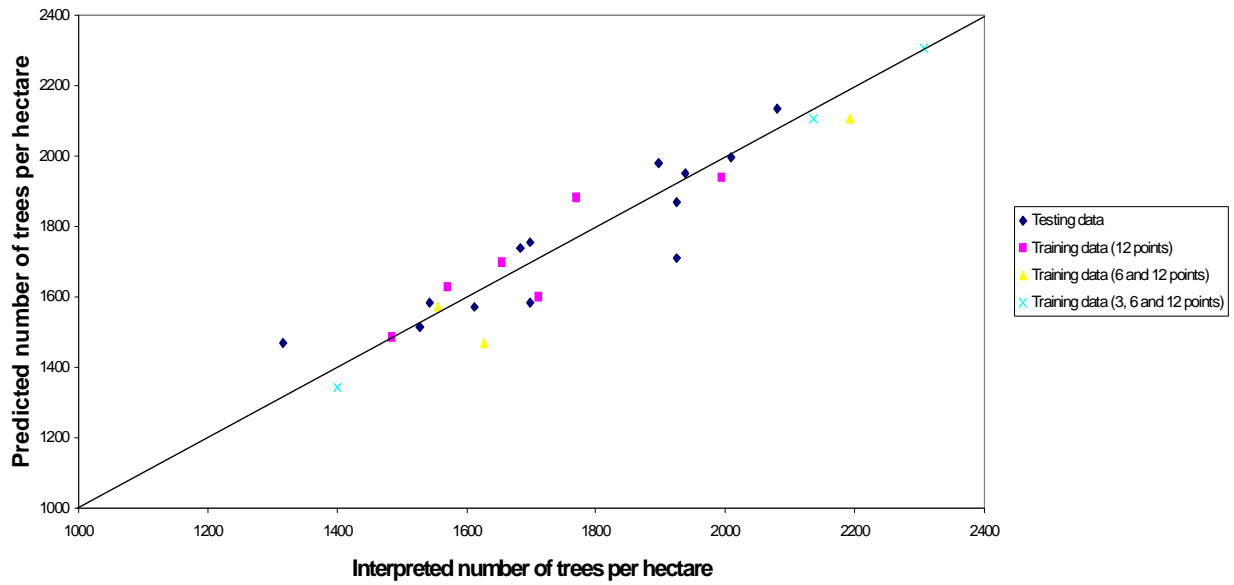
For this analysis, two separate sets of training and testing data were created. Although both are random samples of the population, substantial differences emerge between them.

4.4.2.1 Comparison of tree count results of the two datasets

By examining the results for photointerpreted tree counts and ground counts of trees with $DBH \geq 7$ cm in Table 16, it can be seen that the correlation coefficient is higher and RMSE lower for runs conducted using training and testing set 1 than for runs using training and testing set 2. For the results based on photointerpreted tree counts, Figure 24 shows that for runs in which only three samples were used for training, the samples in training and testing set 1 are much more spread out than the ones in training and testing set 2, giving the program exposure to a wide range of density conditions. The differences for the six and twelve member training and testing sets are smaller, and it is not obvious from Figure 24 why they occur.

For the results obtained using the ground counts of trees with $DBH \geq 7$ cm, the difference in accuracy is probably due to two factors. The first is that the data points selected for runs using only three points were better in training set 1 than in training set 2. This is because the points in training set 2 all have very low tree counts and are not representative of the tree counts found in the points used for testing (Figure 25 and 26). The problem is present in both the photo data and the LiDAR data. The second reason that affects the orthophoto-based results is that training set 2 includes outliers as testing data, which causes the calculated accuracy to decrease (Figure 25). In training set 1, these points are used as training data. Although using these points as training data would cause the testing accuracy to decrease as well, this error would probably be fairly minor so long as other points exist to balance out these points (i.e., the overall trend would not be lost). The error would probably translate into a worse fitness score during optimization rather than a higher testing RMSE or correlation coefficient.

Training and testing data: Photographs, human tree counts, training set 1



Training and testing points: Photograph data, Photointerpreted counts, Set 2

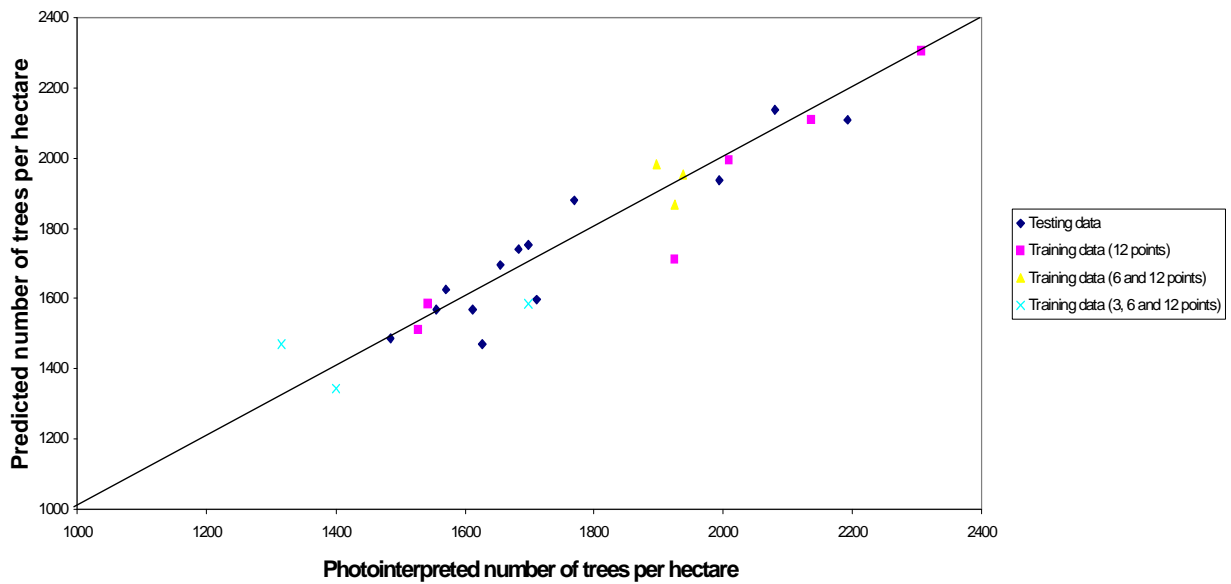
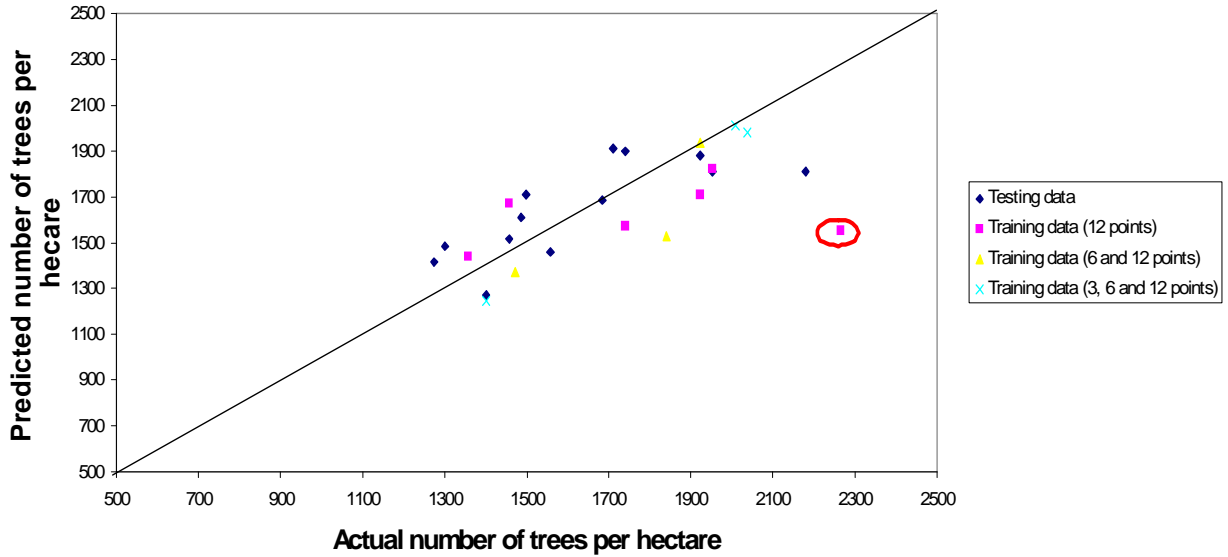


Figure 24. Plots constructed using the photointerpreted tree counts, and predictions of tree counts made when the algorithm was optimized using all 25 plots. The upper plot shows the points used in training set 1 for testing and training using 3, 6 or 12 points. The lower plot shows this information for training set 2.

Training and testing data: Photo dataset using ground based tree counts, DBH > 7cm
Training Set 1



Training and testing data: Photograph data, Ground based tree counts (DBH ≥ 7cm), Training Set 2

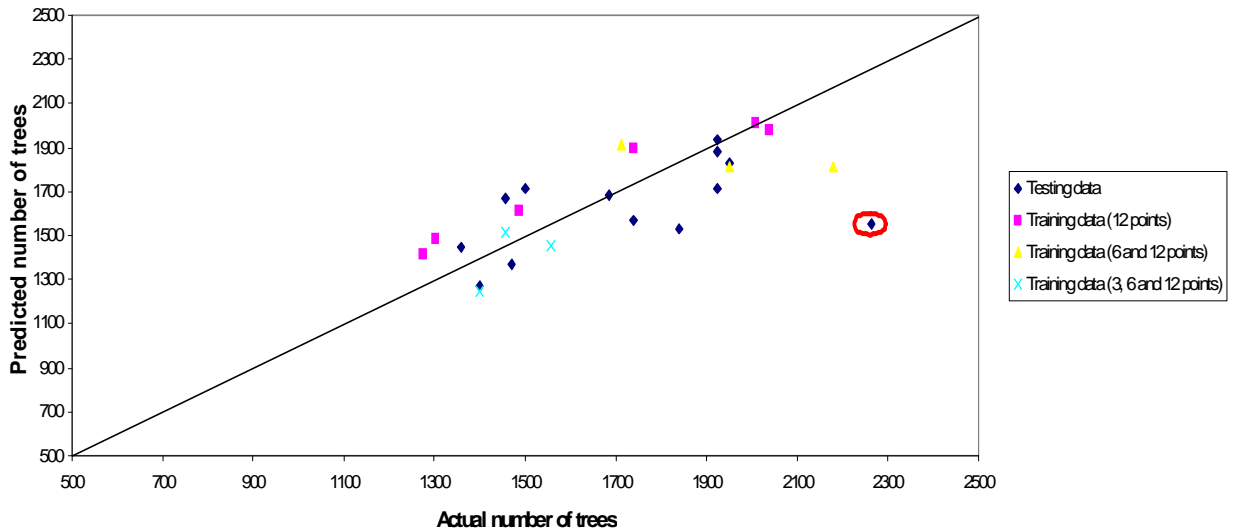
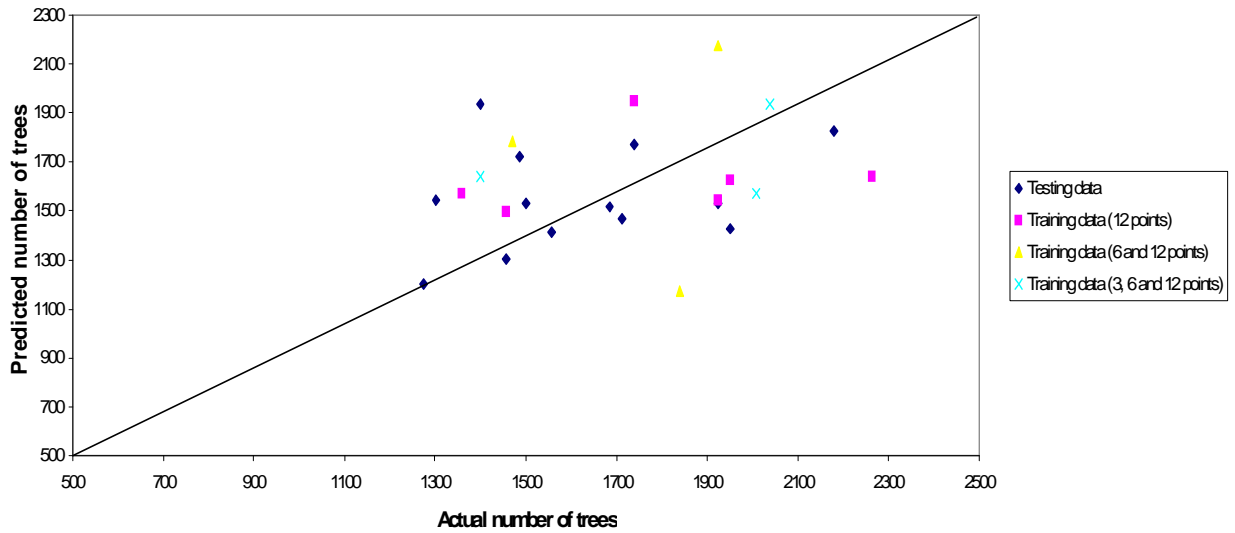


Figure 25. Plots constructed using the ground tree counts of trees with $DBH \geq 7$ cm, and predictions of tree counts made when the algorithm was optimized using all 25 plots and the orthophotograph. The top figure shows training set 1 and the lower figure shows training set 2 (right). An outlying point (standardized residual > 3) has been circled in red.

Training and testing data: Lidar dataset trained using ground counts for trees with DBH > 7cm, Training Set 1



Training and testing data: Lidar dataset trained using ground counts for trees with DBH > 7cm Training Set 2

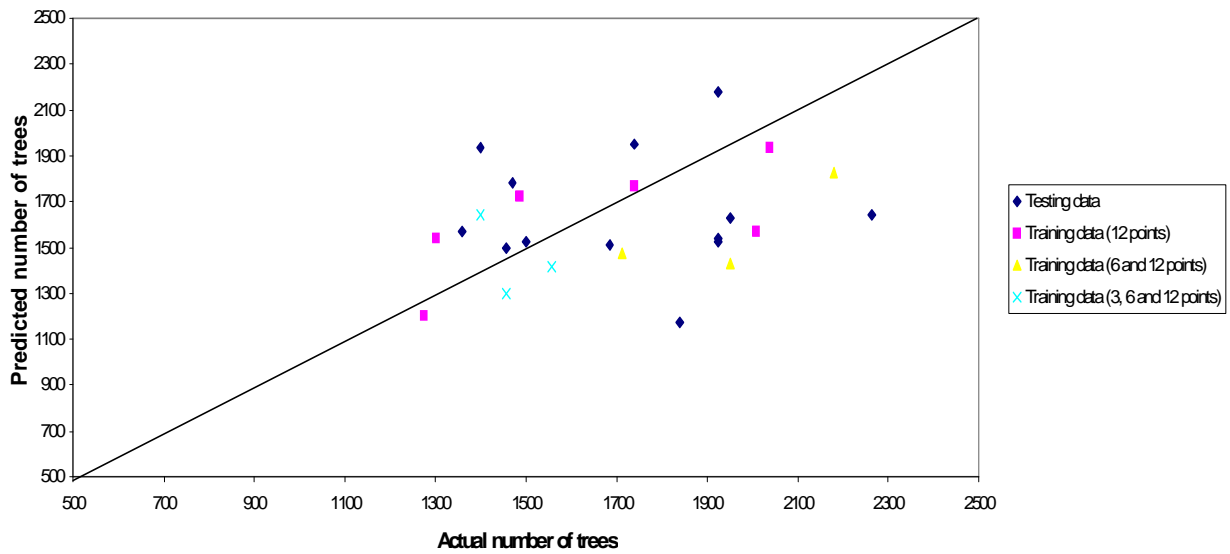


Figure 26. Plots constructed using the ground tree counts of trees with $DBH \geq 7$ cm, and predictions of tree counts made when the algorithm was optimized using all 25 plots and the LiDAR CHM. The upper plot shows training set 1, and the lower plot shows training set 2.

4.4.2.2 Comparison of biomass results for the two datasets

The main difference that is apparent (Table 14) between the biomass results from the two training sets is that when the biomass model is created after optimization has occurred, the correlation coefficient for training set 2 is consistently higher than the correlation coefficient for training set 1. The reason for this can be seen in Figure 23. This figure shows that the points group into two elongate clusters. In training set 2, most of the testing points belong to the cluster that is further to the right, whereas in set 1 they are more evenly distributed between the two clusters.

4.4.3 Evaluation of methods of postprocessing the results

Several techniques for postprocessing the results of the five parameter sets given by the Nelder-Mead simplex were discussed in Section 3.6 and listed in Table 11. Tables 19 and 20 give the correlation coefficients resulting from these techniques, Tables 21 and 22 give the RMSE values obtained by these coefficients, and Table 23 summarizes the results of the techniques. Qualitatively, it appears that the technique in which the tree counts or biomass values were averaged consistently gave higher than average correlation coefficients and lower than average RMSE values for all runs except those in which tree counts were predicted using LiDAR images. The other techniques were much less consistent.

In an effort to quantify this advantage, the difference between the correlation coefficient or RMSE value selected by a technique and the average of all five correlation coefficients or RMSE values for a given run was computed. These values were then averaged for the runs in which three or six plots were used for training (data for runs in which twelve plots were used were not included since not all post processing techniques were applicable in these runs). Table 20 shows these results. Although it can be seen that averaging the five predicted tree count or biomass values produced better average results, a multiple range test using Tukey's W indicated that very few of these improvements were significant ($p \leq 0.05$). The only significant difference was that for the air photo imagery, averaging the five predicted tree counts produced higher correlation coefficients than when one of the five solutions was selected using either 1) the correlation coefficients associated with the training points not (continued on page 133)

Table 19. Correlation coefficients obtained for tree count predictions using the postprocessing techniques described in Table 11. Please refer to Table 11 for an explanation of the abbreviations used. Blank cells indicate tests that the postprocessing technique was not applicable to the data.

Imagery	Predicted data	Training set	Number of training images	Average r	CORv	CORvy	RMSEv	RMSEvy	RMSEy	Av5
Photo	CH	1	3	0.84	0.89	0.89	0.89	0.89	0.86	0.89
Photo	CH	1	6	0.81	0.87	0.84	0.87	0.87	0.70	0.78
Photo	CH	1	12	0.82		0.83			0.83	0.84
Photo	CH	2	3	0.76	0.82	0.82	0.78	0.78	0.62	0.86
Photo	CH	2	6	0.75	0.76	0.76	0.75	0.75	0.75	0.79
Photo	CH	2	12	0.79		0.79			0.79	0.87
Photo	C7	1	3	0.71	0.75	0.75	0.71	0.71	0.69	0.78
Photo	C7	1	6	0.53	0.49	0.60	0.60	0.60	0.49	0.66
Photo	C7	1	12	0.56		0.64			0.64	0.59
Photo	C7	2	3	0.24	0.15	0.15	0.06	0.06	0.21	0.27
Photo	C7	2	6	0.27	0.13	0.32	0.49	0.49	0.32	0.35
Photo	C7	2	12	0.21		0.04			0.16	0.24
Photo	C10	1	3	0.62	0.32	0.70	0.38	0.38	0.51	0.87
Photo	C10	1	6	0.66	0.45	0.71	0.45	0.45	0.45	0.72
Photo	C10	1	12	0.63		0.78			0.78	0.72
Photo	C10	2	3	0.51	0.35	0.35	0.35	0.35	0.35	0.62
Photo	C10	2	6	0.51	0.49	0.49	0.49	0.49	0.47	0.55
Photo	C10	2	12	0.52		0.50			0.50	0.56
LiDAR	C7 (P)	1	3	0.45	0.63	0.63	0.63	0.63	0.44	0.24
LiDAR	C7 (P)	1	6	0.65	0.78	0.78	0.78	0.78	0.78	0.21
LiDAR	C7 (P)	1	12	0.73		0.78			0.78	0.27

Imagery	Predicted data	Training set	Number of training images	Average r	CORv	CORvy	RMSEv	RMSEvy	RMSEy	Av5
LiDAR	C7 (P)	2	3	-0.26	-0.21	-0.21	-0.21	-0.21	-0.21	-0.38
LiDAR	C7 (P)	2	6	-0.06	-0.51	-0.51	-0.51	-0.51	-0.38	-0.07
LiDAR	C7 (P)	2	12	0		-0.09			-0.09	-0.23
LiDAR	C7 (P)	1	3	0.44	0.45	0.45	0.45	0.45	0.45	0.38
LiDAR	C7 (P)	1	6	0.42	0.47	0.47	0.56	0.56	0.56	0.46
LiDAR	C7 (P)	1	12	0.61		0.52			0.52	0.49
LiDAR	C7 (P)	2	3	0.16	-0.07	0.47	-0.07	-0.07	0.48	0.46
LiDAR	C7 (P)	2	6	0.48	-0.11	-0.11	-0.11	0.25	0.43	0.32
LiDAR	C7 (P)	2	12	0.08		0.21			0.21	0.08
LiDAR	C10 (D)	1	3	0.57	0.65	0.59	0.65	0.65	0.49	0.50
LiDAR	C10 (D)	1	6	0.62	0.60	0.60	0.60	0.60	0.61	0.42
LiDAR	C10 (D)	1	12	0.63		0.61		0.61		0.65
LiDAR	C10 (D)	2	3	0.75	0.68	0.68	0.76	0.76	0.68	0.38
LiDAR	C10 (D)	2	6	0.69	0.69	0.69	0.69	0.69	0.69	0.48
LiDAR	C10 (D)	2	12	0.54		0.15		0.15		0.07

Table 20. Correlation coefficients obtained for LiDAR derived biomass predictions using the postprocessing techniques described in Table 11. Please refer to Table 11 for an explanation of the abbreviations used. Blank cells indicate tests that the postprocessing technique was not applicable to the data.

Processing method	Training set	Number of training images	Average correlation	COR _v	COR _{vy}	RMS _v	RMS _{vy}	RMS _y	Av5
P	1	3	0.66	0.65	0.65	0.65	0.65	0.65	0.66
P	1	6	0.64	0.66	0.66	0.66	0.66	0.59	0.65
P	1	12	0.67		0.66			0.66	0.67
P	2	3	0.80	0.79	0.79	0.79	0.79	0.79	0.80
P	2	6	0.79	0.79	0.79	0.79	0.79	0.78	0.79
P	2	12	0.78		0.78			0.78	0.78
D	1	3	0.57	0.65	0.59	0.65	0.65	0.49	0.68
D	1	6	0.62	0.60	0.60	0.60	0.60	0.61	0.62
D	1	12	0.63		0.61		0.61		0.63
D	2	3	0.75	0.68	0.68	0.76	0.76	0.68	0.76
D	2	6	0.69	0.69	0.69	0.69	0.69	0.69	0.69
D	2	12	0.41		0.15		0.15		0.42

Table 21. RMSE values (in trees / ha) obtained for tree count predictions using the postprocessing techniques described in Table 11. Please refer to Table 11 for an explanation of the abbreviations used. Blank cells indicate tests that the postprocessing technique was not applicable to the data.

Imagery	Predicted data	Training set	Number of training images	Average RMSE	CORv	CORvy	RMSEv	RMSEvy	RMSEy	Av5
Photo	CH	1	3	135.8	107.5	107.5	107.5	107.5	148.5	111.8
Photo	CH	1	6	134.4	116.0	130.2	116.0	116.0	210.8	127.3
Photo	CH	1	12	138.6		133.0			133.0	142.9
Photo	CH	2	3	246.2	181.1	181.1	169.8	169.8	451.3	150.0
Photo	CH	2	6	175.4	164.1	164.1	161.3	161.3	161.3	127.3
Photo	CH	2	12	183.9		178.3			178.3	150.0
Photo	C7	1	3	216.5	210.8	210.8	233.4	233.4	222.1	200.9
Photo	C7	1	6	280.1	256.1	291.4	291.4	291.4	256.1	237.7
Photo	C7	1	12	288.6		266.0			266.0	278.7
Photo	C7	2	3	329.6	328.2	328.2	332.5	332.5	314.1	297.1
Photo	C7	2	6	353.7	339.5	210.8	263.1	263.1	210.8	299.9
Photo	C7	2	12	326.8		353.7			164.1	345.2
Photo	C10	1	3	316.9	502.2	285.8	318.3	318.3	393.3	241.9
Photo	C10	1	6	288.6	339.5	274.5	339.5	339.5	339.5	278.7
Photo	C10	1	12	277.3		237.7			237.7	263.1
Photo	C10	2	3	297.1	299.9	299.9	299.9	299.9	299.9	251.8
Photo	C10	2	6	240.5	377.7	377.7	377.7	377.7	254.6	233.4
Photo	C10	2	12	278.7		291.4			291.4	260.3
LiDAR	C7 (P)	1	3	400.4	469.7	469.7	469.7	469.7	333.9	434.3
LiDAR	C7 (P)	1	6	377.7	527.7	527.7	527.7	527.7	527.7	355.1
LiDAR	C7 (P)	1	12	345.2		445.6			445.6	352.3
LiDAR	C7 (P)	2	3	502.2	462.6	462.6	462.6	462.6	462.6	389.0
LiDAR	C7 (P)	2	6	589.9	537.6	537.6	537.6	537.6	516.4	427.2
LiDAR	C7 (P)	2	12	445.6		421.6			421.6	356.5
LiDAR	C7 (P)	1	3	343.8	328.2	328.2	328.2	328.2	328.2	338.1
LiDAR	C7 (P)	1	6	340.9	481.0	481.0	241.9	241.9	241.9	623.9
LiDAR	C7 (P)	1	12	251.8		266.0			266.0	250.4
LiDAR	C7 (P)	2	3	465.4	367.8	754.0	367.8	367.8	1090.7	632.4
LiDAR	C7 (P)	2	6	629.5	568.7	568.7	568.7	321.1	297.1	339.5
LiDAR	C7 (P)	2	12	384.8		319.7			319.7	356.5
LiDAR	C10 (D)	1	3	343.8	352.3	352.3	333.9	333.9	345.2	298.5
LiDAR	C10 (D)	1	6	340.9	342.4	342.4	274.5	274.5	335.3	271.6
LiDAR	C10 (D)	1	12	251.8		222.1			249.0	224.9
LiDAR	C10 (D)	2	3	465.4	360.8	360.8	384.8	384.8	338.1	355.1
LiDAR	C10 (D)	2	6	629.5	640.9	640.9	602.7	602.7	616.8	588.5
LiDAR	C10 (D)	2	12	384.8		376.3			376.3	350.8

Table 22. RMSE values (in t / ha) obtained for LiDAR-based biomass predictions using the postprocessing techniques described in Table 11. Please refer to Table 11 for an explanation of the abbreviations used. Blank cells indicate tests that the postprocessing technique was not applicable to the data.

Processing method	Training set	Number of training images	Average RMSE	CORv	CORvy	RMSv	RMSvy	RMSy	Av5
P	1	3	22.4	17.1	66.1	17.1	17.1	57.4	22.4
P	1	6	18.2	19.8	19.8	19.8	19.8	19.2	18.0
P	1	12	18.5	0.0	19.8	0.0	19.8	19.8	18.5
P	2	3	56.9	68.8	68.8	48.1	48.1	68.8	55.7
P	2	6	23.3	27.3	27.3	27.9	27.9	27.0	22.4
P	2	12	15.8	0.0	22.9	0.0	22.9	22.9	15.8
D	1	3	60.0	17.1	66.1	17.1	17.1	57.4	67.9
D	1	6	19.0	19.8	19.8	19.8	19.8	19.2	18.8
D	1	12	19.5	0.0	19.8	0.0	19.8	19.8	19.5
D	2	3	56.0	68.8	68.8	48.1	48.1	68.8	54.3
D	2	6	27.6	27.3	27.3	27.9	27.9	27.0	27.6
D	2	12	20.7	0.0	22.9	0.0	22.9	22.9	20.5

Table 23. Overall effectiveness of the postprocessing methods in terms of correlation coefficient (Δr) and RMSE ($\Delta RMSE$). Please see Section 3.3.3 for a description of the method used to construct this chart. Results that show that a postprocessing technique was effective are for a given set of data are depicted in italics. For each row, the best value is noted with an asterisk.

Imagery	Predicted quantity	Effectiveness measure	CORv	CORvy	RMSv	RMSvy	RMSy	Av5
Photo	Tree count	Δr	-0.06	<i>0.04</i>	-0.02	-0.02	-0.06	<i>0.08*</i>
Photo	Tree count	$\Delta RMSE$	17.40	<i>-12.90</i>	<i>-0.300</i>	<i>-0.300</i>	20.80	<i>-38.2*</i>
LiDAR	Tree count	Δr	-0.10	-0.05	-0.09	-0.06	<i>0.01*</i>	-0.05
LiDAR	Tree count	$\Delta RMSE$	0.7	32.80	<i>-27.40</i>	<i>-48.1*</i>	0.3	<i>-31.3</i>
LiDAR	Biomass	Δr	0.00	-0.01	<i>0.01</i>	<i>0.01</i>	-0.03	<i>0.02*</i>
LiDAR	Biomass	$\Delta RMSE$	<i>-2.1</i>	10.0	<i>-7.1*</i>	<i>-7.1*</i>	7.6	0.6

used in the optimization, or 2) the RMSE of the training points used in the optimization. Although the superiority of averaging the results of the five parameters could be only partially shown using the quantitative technique, based on the qualitative analysis it appears that it is a worthwhile procedure and gives somewhat better results than the other techniques.

4.5 Comparison with results from other studies

Comparing the results of the current study to the results obtained by other researchers is a task that cannot be done well. This the case for a number of reasons:

- All prior studies except Popescu *et al.* (2003) and Popescu and Wynne (in press) used different study sites, often having markedly different forest types on them.
- The imagery used in this study differed from the imagery used in most other studies.
- For the studies that used computer vision techniques to locate trees in aerial photographs, different evaluation techniques were often used than the ones used in this study. In most cases, the difference in evaluation technique is due to different data being collected at the study sites. For example, Pollock (1994) , Brandtberg and Walter (1999), and Larsen (1999) collected ground data on the locations of every tree in their study plots, and used these data to calculate the percentage of trees found by their programs. In this study, the locations and stand densities of plots were determined instead. This precluded a calculation of the percentage of trees found by the program that correspond to trees on the ground.

Despite these difficulties, some comparisons can be made.

4.5.1 Comparison of tree location and density results

The results of past studies that examined tree location and density can be divided into two groups: Studies that used precise ground locations of trees, and those that did not. The study described in this dissertation did not use precise ground locations of trees. In order to make a comparison with these studies, it is therefore necessary to use the statistic on the fraction of trees found by the program that correspond to trees found by a human photointerpreter (89%). The

accuracies obtained by studies using measured tree locations are:

- Brandtberg and Walter (1999): 54% of trees located
- Larsen (1999): 95% of trees located
- Pollock (1994): 57% of trees located
- Wulder *et al.* (2000): 5 to 80% of the trees located

Although the success rate of the program described in this dissertation exceeds the success rate of all but one of these studies, using locations obtained by a photointerpreter is not a very good surrogate for ground locations. This is because the human photointerpreter is likely to make the same mistakes as the program (e.g., missing small trees), leading to a higher accuracy statistic.

Three studies used ground density measurements rather than tree locations for accuracy assessment: Gougeon (1995), Dralle and Rudimo (1996), Popescu *et al.* (2003), and Popescu and Wynne (in press) and . Gougeon (1995) was able to obtain results that were within 7.7% of ground counts, and 71% of the trees located by his algorithm corresponded to trees found by a human interpreter. The accuracy of the ground counts is greater than the accuracy obtained using the algorithm described in this study. However, the accuracy was assessed using larger assessment areas than the individual plots used in this dissertation, so the results are difficult to compare since errors would tend to average out in the larger plots. The fraction of trees found by Gougeon's (1995) program that correspond to trees found by a human interpreter is lower than the fraction found by the program in this study.

The algorithm developed by Dralle and Rudimo (1996) predicted densities at the subplot level that were between -6.9 and 5.0% of the true values. Although larger errors are present in the predictions obtained for this dissertation (even when all twenty five plots were used for training), the two studies are not very comparable. Dralle and Rudimo (1996) worked with much higher resolution data (15cm rather than 50cm), and the stands they used were all the same age and were much older (48 years old) and therefore probably contained larger trees and larger gaps among the trees than the stands used in the current study.

Popescu *et al.* (2003) and Popescu and Wynne (in press) found r^2 values for tree counts in softwood stands between 0.1548 and 0.2626 using LiDAR data. Depending on the training and testing data used, the r^2 values for this study ranged between 0 and 0.53. Since different sites and

data collection procedures were used, it is not possible to say with confidence which LiDAR-based technique performed better.

Overall, the results of this study appear to compare relatively well to other comparable studies both in terms of locating individual trees and predicting density. However, large site and image differences make this comparison difficult to make.

4.5.2 Comparison of biomass results

The biomass results from this study are comparable to the results obtained in other studies. When all the plots were used for training and testing, r^2 values between 0.49 and 0.52, and RMSEs between 13.77 and 14.20 t / ha were achieved. Much more variability was present when subsets of the data were used for training and testing. For these runs, the r^2 values ranged between 0.02 and 0.64, and the RMSE values ranged between 17.16 and 81.93 t / ha. In nearly all cases the r^2 values were significant at the 95% or 99% level based on a one tailed test. Table 24 shows an abridged version of Table 2, in which only studies that used remote sensing techniques to determine the aboveground biomass of stands are considered.

With the exception of the study by Hussin *et al.* (1991), the RMSE values obtained by this study are either better than or are comparable to the other studies listed. This indicates that the program developed in this study is able to predict biomass at least as precisely as other studies, suggesting that the program is quite sensitive to differences in forest structure that affect biomass.

Although the r^2 values of the stands are lower than the majority of the results, the lower r^2 values may result from the lack of variability in the dataset rather than poor algorithm design since only one forest type was considered (loblolly pine plantations) with a very narrow range of ages (eleven to sixteen years). Other studies included a much wider range of forest. For example, Lefsky *et al.* (1999a) included stands ranging from very young to old growth, and Steininger (2000) includes stands ranging from four to thirty years of age that have a wide range of site indices due to differences in prior land use. Having a wide range of biomass values reduces the need for individual measurements to be precise in order to have a high r^2 .

With the current study site and imagery, it is unlikely that the r^2 value can be substantially improved since the precision is already very high compared to the other studies and both the study

site and imagery are limiting. The study site is limiting because of the substantial number of subdominant trees present due to extensive management that may or may not be visible depending on their position in the canopy and the chance positioning of the laser beam. The imagery is limiting due to the density of the LiDAR which did not permit young stands with low biomass to be consistently identified because these trees were often missed by the laser beam due to their small size. In future studies, it may be possible to raise the r^2 values to those of the other studies by choosing a study site which has been intensively managed. This would enable older stands that were excluded from this study due to the very high number of subdominant trees to be included, thereby increasing the range of the biomass values. A higher point spacing may also represent an improvement by allowing younger stands to be included and by possibly allowing better identification of trees in the stands used in the present study. It may also be possible to use a non-object-oriented approach that does not rely on the identification of individual trees and possibly includes the use of intermediate returns, since this would permit the inclusion of other in which individual trees that cannot be accurately identified. However, doing this would forgo the benefits described in Section 1.3.6.2.

In summary, the biomass accuracies obtained in this study compare well to those obtained in other studies. The RMSE values were lower than those obtained by all but one comparable study. Although the correlation between actual and predicted biomass was lower for this study than in other studies, most correlations were highly significant, and the low correlation is likely to be due to the nature of the study site rather than algorithm performance.

4.6 Comparison of the Nelder-Mead simplex and genetic algorithm techniques

The Nelder-Mead simplex algorithm and the genetic algorithm techniques were compared using the procedure described in Section 3.6. Figures 27, 28, and 29 show the results of this procedure. From Figure 27, it appears that the Nelder-Mead simplex technique is somewhat faster than the genetic algorithm when only three plots are used for training. The other plots (Figures 28 and 29) do not indicate a clear trend, either in speed or in the final fitness score achieved. These results indicate that while the genetic algorithm performed well, it does not offer a benefit over the
(Continued on page 144)

Table 24. Studies that used remote sensing techniques to determine the aboveground biomass of forest stands. Abridged from Table 2.

Study	Imagery type	Forest type	r ²	RMSE (t / ha)
Wu (1987)	Radar	Pine plantations	0.83	30.52
Hussin <i>et al.</i> (1991)	Radar	Pine plantations	0.977	0.0654
LeTuon <i>et al.</i> (1992)	Radar	Pine plantations	0.95	Not reported
Lefsky <i>et al.</i> (1999a)	Large footprint LiDAR	Douglas-fir / western hemlock	0.91 (adj.)	Not reported
Lefsky <i>et al.</i> (1999b)	Large footprint LiDAR	Deciduous forest	0.8	Not reported
Lefsky <i>et al.</i> (2001)	Large footprint LiDAR	Douglas-fir / western hemlock	0.86 (adj.)	163.6
Drake <i>et al.</i> (2002)	Tropical wet forest	Tropical wet forest	0.73	60.02
Nelson <i>et al.</i> (1988)	Small footprint LiDAR	Pines (including plantations), hardwoods	0.55	67.7
Popescu <i>et al.</i> (2003), Popescu and Wynne (in press)	Small footprint LiDAR	Pines, hardwoods	0.82 (pines), 0.33 (hardwoods)	29 (pines), 44 (hardwoods)
Shugart <i>et al.</i> (2000)	Air photos / high resolution optical satellite	Boreal	0.939	19.94
Lefsky <i>et al.</i> (2001)	High resolution optical	Douglas-fir / western hemlock	0.47	204.5
Steininger (2001)	Medium resolution optical	Tropical secondary forest	0.701	Not reported

Study	Imagery type	Forest type	r²	RMSE (t / ha)
Lefsky <i>et al.</i> (2002)	Medium resolution optical	Douglas-fir / western hemlock	0.31 (single date, adj.), 0.60 (multiple date, adj.)	320.3 (single date), 238.5 (multiple date)
Lefsky <i>et al.</i> (2001)	Medium resolution hyperspectral	Douglas-fir / western hemlock	0.38 (adj.)	299.9

Optimization Technique Comparison: 3 Training Plots

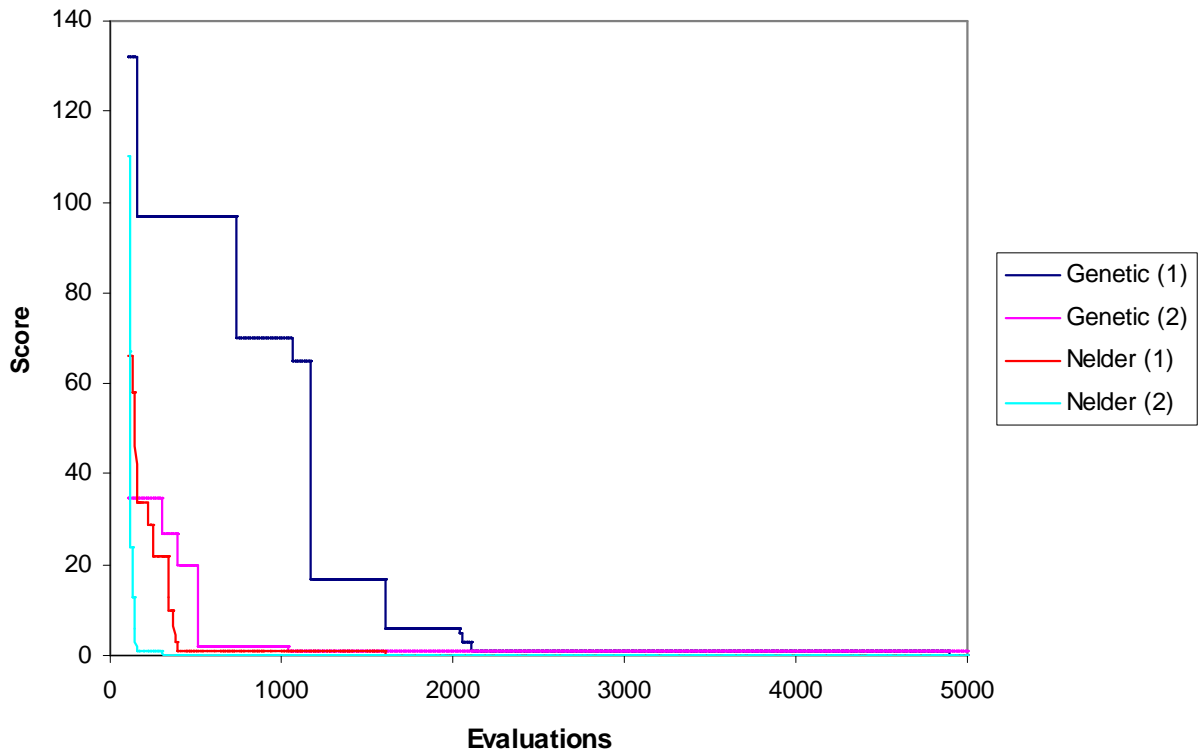


Figure 27. Comparison of the Nelder-Mead simplex and genetic algorithm optimization techniques. Optimizations were run for 5000 evaluations using photographs and ground count data ($DBH \geq 7$ cm) from three plots.

Optimization Technique Comparison: 12 Training Plots

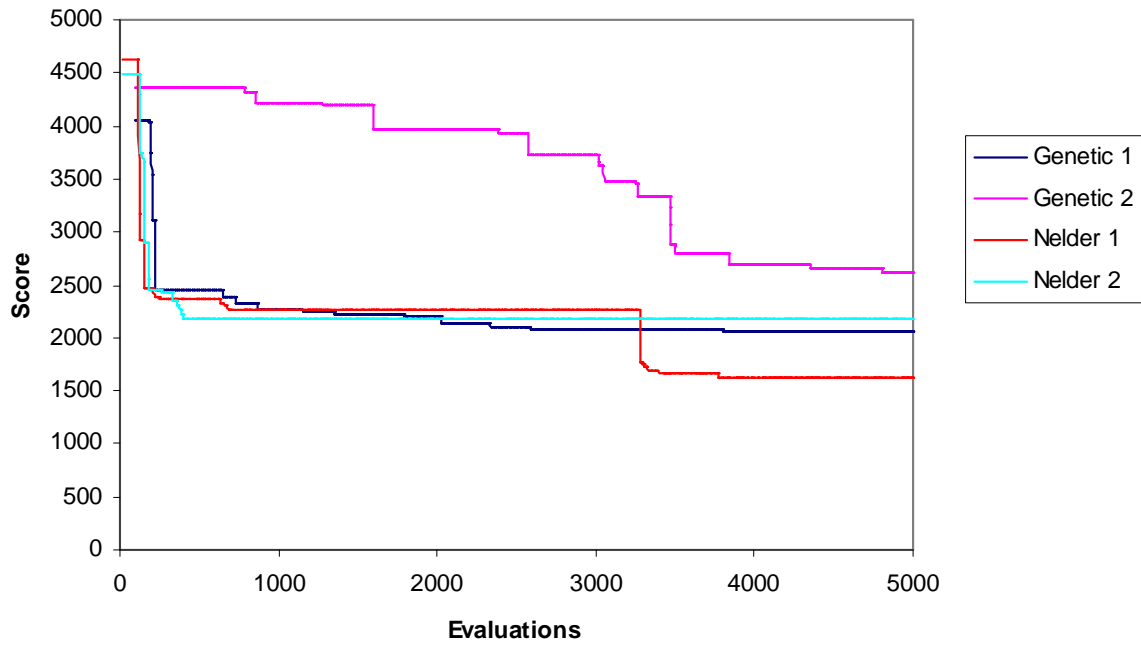


Figure 28. Comparison of the Nelder-Mead simplex and genetic algorithm optimization techniques. Optimizations were run for 5000 evaluations using photographs and ground count data ($DBH \geq 7$ cm) from twelve plots.

Optimization Technique Comparison: 25 Training Plots

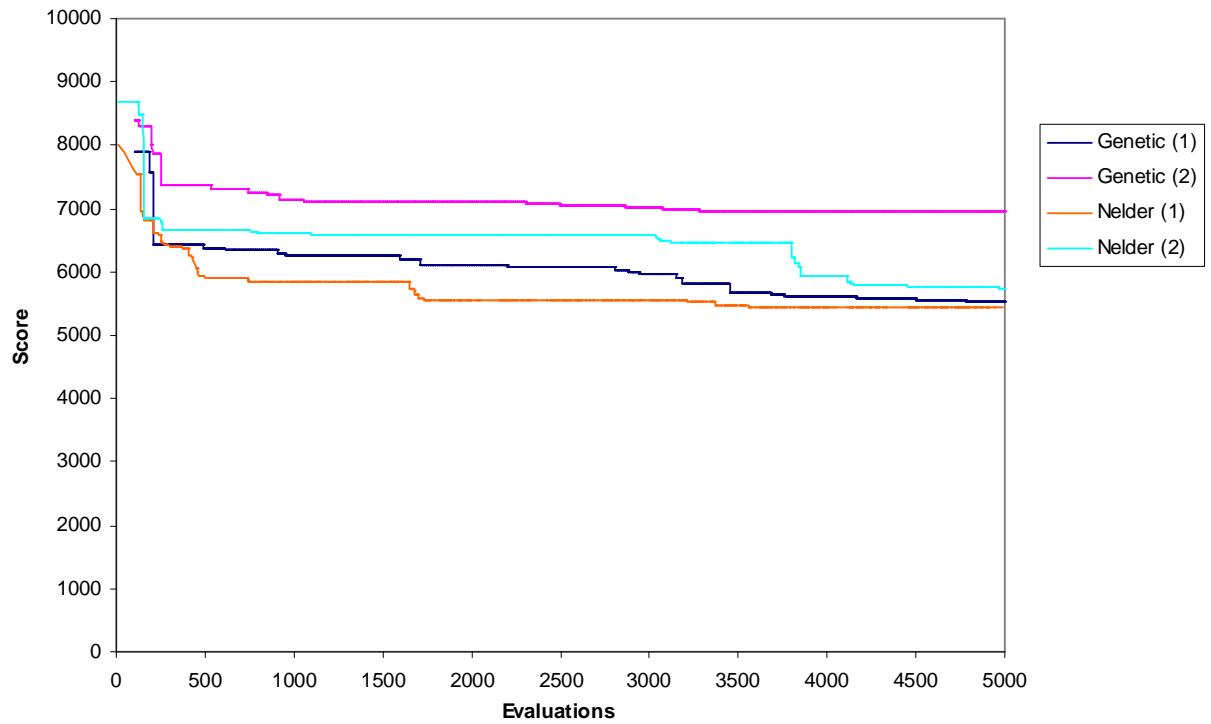


Figure 29. Comparison of the Nelder-Mead simplex and genetic algorithm optimization techniques. Optimizations were run for 5000 evaluations using photographs and ground count data ($DBH \geq 7$ cm) from 25 plots.

Nelder-Mead simplex algorithm. Therefore, it can be concluded that a global optimization technique is not required to set the parameters for the computer vision algorithm. This suggests that in this case local maxima are relatively insignificant, although they may be more significant when the tree recognition algorithm is applied to other study sites.

Chapter 5

Conclusions

5.1 Summary and discussion of results

Rising carbon dioxide levels have the potential to adversely affect the earth and the quality of life it is able to offer its inhabitants. Remote sensing can help researchers and forest managers understand the roles forests play in the carbon cycle by producing biomass values that can be readily converted to the mass of carbon stored by a forest, and forest parameters that can be input into forest process models. Individual tree-based remote sensing techniques using high spatial resolution imagery are an excellent choice for many forestry applications for a variety of reasons. These reasons include the resistance of this approach to edge effects and minor illumination and forest differences.

The primary objective of this dissertation was to engineer and implement the first individual tree-based algorithm for determining forest biomass and density using high spatial resolution imagery that uses parameters set using training data. The objective was successfully realized; the description of the algorithm that was developed is described in Section 3.2, and the programs written to implement the algorithm are given in Appendices A - F.

The secondary objective was to perform a first test of the algorithm using data from young loblolly pine plantations in the Appomattox-Buckingham State Forest. Generally the results from the program were promising, since the results for both density and biomass often resulted in highly significant ($p \leq 0.05$ or $p \leq 0.01$) correlations and low RMSE values. However, the results did vary considerably and depended heavily on the training and testing sets used.

The most successful prediction for tree counts occurred when the training and testing data consisted of tree counts obtained by a human photointerpreter together with subsets of the orthophotograph. In this case, r^2 values as high as 0.88 and RMSE values as favorable as 83 trees / ha were obtained, and 89% of the trees found by the program corresponded to trees found by the

human photointerpreter. This suggests that the program was able to closely mimic the way in which humans interpret orthophotographs.

The results obtained using ground based counts were generally less accurate, but most correlations were significant at the 95% or 99% level, and r^2 values as high as 0.81 for orthophotograph data / 0.61 for LiDAR data, and RMSE values as low as 156 trees per hectare for orthophotograph data / 232 trees per hectare for LiDAR data were obtained. In all cases the orthophotograph data produced better tree count estimates than the LiDAR data. Generally density estimates for trees with $DBH \geq 10\text{cm}$ were better than estimates for trees with $DBH \geq 7\text{cm}$. It is also noteworthy that when trained using the entire set of plots as training data the program was able to obtain more accurate predictions of ground based tree counts than the human photointerpreter.

The biomass results obtained using the ground based data were relatively precise, with r^2 values up to 0.64 and RMSE values as favorable as 15.6 t / ha. Although the r^2 values were lower than the r^2 values found in many other studies, the RMSE values were better than those obtained by the majority of prior attempts and nearly all the correlations were significant at the 95% or 99% level. Two methods of determining biomass were evaluated (developing biomass prediction models after and during optimization) and surprisingly both methods performed equally well, suggesting that it is better to develop the model after optimization since less training time is required.

The Nelder-Mead simplex algorithm used for the majority of optimizations was designed to produce five sets of output parameters based on five starting points. Several methods were evaluated for merging these parameter sets or selecting the best set. It was observed that averaging the predicted tree counts or biomass values obtained using the five parameter sets produced better results than the other techniques, but this was only partially proven statistically.

A comparison was conducted between the Nelder-Mead simplex optimization technique and a genetic algorithm in order to determine whether there was any benefit in using a global optimization technique like the genetic algorithm. Although the genetic algorithm performed well, it was not consistently faster than the Nelder-Mead simplex technique, nor did it produce

consistently more optimal solutions. It is therefore concluded that a global optimization technique is not required for this application.

Overall these results indicate that the algorithm passed its first test since it was able to accurately determine stand density and biomass using orthophotographs and small footprint LiDAR imagery. This is especially significant considering that the imagery used for this study had a coarser resolution than the imagery used for many other similar studies (see Section 2.3), and the forest was extensively managed.

5.2 Caveats

When considering the results of this study, one must keep in mind the limitations that the data presented that were not present in other similar studies. Factors that made this study challenging to conduct include:

- The resolution of the imagery was coarser than that of the imagery used in most of the other object oriented studies described in Section 2.2.
- A narrow range of ages and management techniques was examined. This meant that the variance in the biomass was lower than in other studies, resulting in low correlation coefficients.
- The stands were managed extensively, unlike the stands used in several other research efforts.
- The GPS was probably imprecise, leading to a mismatch between the ground data and the imagery.
- There was a four to five year temporal mismatch between the ground data and the orthophotograph data. During this time period uneven biomass accumulation and mortality may have taken place.

5.3 Future research

Since the algorithm passed its first test, additional testing and algorithm refinements are merited. There are many possible future studies that would be both informative and interesting. Among the most interesting would be to see whether conducting the study with higher spatial resolution imagery and more intensively managed stands would yield better results. It is expected that it would for many reasons (e.g., the greater ability to resolve small trees, and the absence of

understory trees and large gaps), but it is not possible to properly evaluate this prediction using the current imagery and study site.

It would also be useful to examine whether the algorithm meets its goal of working with multiple image and forest types. This could be done by acquiring images and ground data collected in different areas with different forest types and re-running the analysis. Due to its widespread availability, an examination of results obtained using Quickbird data would be especially interesting.

The optimization and postprocessing techniques may also be a fruitful area of research. Logical extensions of the techniques used in this dissertation would be to average the results from more than five sets of parameters obtained from a single set of training data, and to run the optimization using different subsets of the training data, and then to average the resulting predictions.

A possible future extension of the computer vision portion of the algorithm would be to enable it to operate at the subpixel level. This may permit better crown boundaries to be more precisely defined, and for greater differences in crown width to be detected.

References

- Aber, J. D. and C. A. Federer. 1992. A generalized, lumped-parameter model of photosynthesis, evapotranspiration and net primary productivity in temperate and boreal forest ecosystems. **Oecologia** 92: 463-474.
- Agbanhan, J., M. Minami. and T. Asakura. 2000. A model-based real-world scene recognition using local/global search of a GA for manipulator real-time visual servoing. In **IECON 2000**, Nagoya, Japan, October 20-28, 2000. p. 2195-2200.
- Alemdag, I. S.. 1986. Estimating oven-dry mass of trembling aspen and white birch using measurements from aerial photographs. **Canadian Journal of Forest Research** 16: 163-165.
- Avery, G.. 1958. Composite aerial volume table for southern pines and hardwoods. **Journal of Forestry** 56: 741-745.
- Baldwin (Jr.), V. C., K. D. Peterson, H. E. Burkhart, R. L. Amateis, and P. M. Dougherty. 1997. Equations for estimating loblolly pine branch and foliage weight and surface area distributions. **Canadian Journal of Forest Research** 27: 918-927.
- Bhanu, B., S. Lee and S. Das. 1995. Adaptive image segmentation using genetic and hybrid search methods. **IEEE Transactions on Aerospace and Electronic Systems** 31:1268-1291.
- Boscolo, M., M. Powell, M. Delany, S. Brown, and R. Faris. 2000. The cost of inventorying and monitoring carbon: Lessons from the Noel Kempff Climate Action Project. **Journal of Forestry** (98:9): 24-27, 29-31.

Bragg, D. C.. 2001. A local basal area adjustment for crown width prediction. **Northern Journal of Applied Forestry** 18(1): 22 - 28.

Brandtberg, T. and F. Walter. 1999. An algorithm for delineation of individual tree crowns in high spatial resolution aerial images using curved edge segments at multiple scales. In: D. A. Hill and D. G. Leckie (editors) **Automated Interpretation of High Spatial Resolution Digital Imagery for Forestry**. Victoria, British Columbia, Canada, February 10-12, 1998. Natural Resources Canada, Canadian Forest Service, Pacific Forestry Centre, Victoria, British Columbia. p. 41-54.

Breuer, G.. (1979) Can forest policy contribute to solving the CO2 problem? **Environment International** 2: 449-451.

Brown, S., O. Masera, and J. Sathaye. 2000. Project based activities. In: R. T. Watson, I. R. Noble, B. Bolin, N. H. Ravindranath, D. J. Verardo, and D. J. Dokken (editors) **Land-use, Land-use Change, and Forestry**. New York: Cambridge University Press. p. 283-228.

Cain, M. D., and M. G. Shelton. 2003. Effects of alternative thinning regimes and prescribed burning in natural, even-aged loblolly-shortleaf pine stands: 25 year results. **Journal of Applied Forestry** 27: 18 - 29.

Charbonneau, P.. 2002. An introduction to genetic algorithms for numerical optimization. **NCAR Technical Note** TN-450. 64p.

Clark (III), A. and M. A. Taras. 1976. Comparison of aboveground biomass of the four major southern pines. **Forest Products Journal** 26(10): 25-29.

Clutter, J. L., J. C. Fortson, L. V. Pienaar, G. H. Brister, and R. L. Bailey. 1983. **Timber Management: A Quantitative Approach**. Malabar, FL: Krieger Publishing. 333p.

Cohen, W. B., M. E. Harmon, D. O. Wallin, and M. Fiorella. 1996. Two decades of carbon flux from forests of the Pacific Northwest: Estimates from a new modelling strategy. **BioScience** 46: 836-844.

Cooper C.F.. 1983. Carbon storage in managed forests. **Canadian Journal of Forest Research** 13: 155-166.

Corvallis Microtechnology. 2004. March-II-E. <http://www.cmtinc.com/fieldcmp/march.html>.

Culvenor, D. S., N. Coops, R. Preston and K. G. Tolhurst. 1999. A spatial clustering approach to automated tree crown delineation. In: D. A. Hill and D. G. Leckie (editors) **Automated Interpretation of High Spatial Resolution Digital Imagery for Forestry**. Victoria, British Columbia, Canada, February 10-12, 1998. Natural Resources Canada, Canadian Forest Service, Pacific Forestry Centre, Victoria, British Columbia. p. 67-80.

Czaplewski, R.. 1998. Use of Advanced Remote Sensing Technologies for Annual State Inventories. USDA Forest Service, Forest Inventory and Analysis Program, Unpublished Report. <http://www.fs.fed.us/rm/ftcol/rwu4804/farm2.htm>.

Davis, M. (1991). **Handbook of Genetic Algorithms**. New York: Van Nostrand Reinhold. 384p..

DigitalGlobe. 2003. **Quickbird Imagery Products Commercial Pricing**. <http://www.digitalglobe.com/downloads/QuickBird%20Imagery%20Products%20-%20Price%20Table.pdf>.

Dobson, M. C., L. E. Pierce, K. Sarabandi, F. T. Ulaby, and T. Sharik. 1992. Preliminary analysis of ERS-1 SAR for forest ecosystem studies. **IEEE Transactions of Geoscience and Remote Sensing** 30: 203- 211.

Dobson, M. C., F. T. Ulaby, L. E. Pierce, T. L. Sharik, K. M. Bergen, J. Kellndorfer, J. R. Kendra, E. Li, Y. C. Lin, A. Nashashibi, K. Sarabandi, and P. Siqueira. 1995. Estimation of forest biophysical characteristics in Northern Michigan with SIR-C/X-SAR. **IEEE Transactions on Geoscience and Remote Sensing** 33: 877-895.

Drake, J.B., R.O. Dubayah, D.B. Clark, R.G. Knox, J.B. Blair, M.A. Hofton, R.L. Chazdon, J.F. Weishampel, and S. Prince. 2002. Estimation of tropical forest structural characteristics using large-footprint lidar. **Remote Sensing of Environment** 79: 305-319.

Dralle, K. and M. Rudemo. 1996. Stem number estimation by kernel smoothing of aerial photos. **Canadian Journal of Remote Sensing** 26: 1228-1236.

Dubayah, R.O. and J.B. Drake. 2000. Lidar remote sensing for forestry. **Journal of Forestry** 98: 44-46.

Dyson, F. J.. 1977. Can we control carbon dioxide in the atmosphere? **Energy** 2: 287-291

Dyson, F.J., and G. Marland. 1979. Technical fixes for the climatic effects of CO₂. In: Elliott, W.P. and L. Machta (eds.) **Workshop on the Global Effects of Carbon Dioxide from Fossil Fuels**, Miami Beach, FL, March 7-11, 1977. U.S. Department of Energy, Washington, DC. p. 111-118.

Edwards, M. B. and W. H. McNab. 1979. Biomass prediction for young southern pines. **Journal of Forestry** 77: 291-292.

Environmental Protection Agency. 2003. **Draft Inventory of U.S. Greenhouse Gas Emissions and Sinks: 1990-2001.**

<http://yosemite.epa.gov/oar/globalwarming.nsf/content/ResourceCenterPublicationsGHGEmissionsUSEmissionsInventory2003.html>. 233p.

Eurimage. 2003. **Eurimage Price List: International.**

http://www.eurimage.com/products/docs/P_S.pdf.

Fiorentini A., G. Baumgartner, S. Magnussen, P. H. Schiller, and J. P. Thomas. 1990. The perception of brightness and darkness. In Spillmann L., & Werner J. S. (editors). **Visual Perception: The Neurophysiological Foundations.** San Diego: Academic Press. p. 129-161.

Franklin, J.. 1986. Thematic Mapper analysis of coniferous forest structure and composition. **International Journal of Remote Sensing** 7: 1287-1301.

Fransson, J. E. S., G. Smith, J. Askne, and H. Olsson. 2001. Stem volume estimation in boreal forests using ERS-1/2 coherence and SPOT XS optical data. **International Journal of Remote Sensing** 22: 2777-2791.

Gates, D. M.. 1993. **Climate Change and Its Biological Consequences.** Sunderland, MA: Sinauer Associates. 280p.

Geneletti, D. and B. G. H. Gorte. 2003. A method for object-oriented land cover classification combining Landsat TM data and aerial photographs. **International Journal of Remote Sensing** 24: 1273 - 1286.

Goldberg, D. E.. 1991. Real-coded genetic algorithms, virtual alphabets and blocking. **Complex Systems** 5: 139-167.

Goodale, C. L., M. J. Apps, R. A. Birdsey, C. B. Field, L. S. Heath, R. A. Houghton, J. C. Jenkins, G. H. Kohlmaier, W. Kurz, S. Liu, G.-J. Nabuurs, S. Nilsson, and A. Z. Shvidenko. 2002. Forest carbon sinks in the northern hemisphere. **Ecological Applications** 12: 891-899.

Gougeon, F. A. (1995). A crown-following approach to the delineation of individual tree crowns in high spatial resolution aerial images. **Canadian Journal of Remote Sensing** 21: 274-284.

Harmon, M. E. and J. B. Domingo. 2001. A Users Guide to STANDCARB version 2.0: A model to simulate the carbon stores in forest stands.

<http://www.fsl.orst/iter/pubs/models/Standcarb2.htm>.

Hepp, T. E. and G. H. Bristner. 1982. Estimating crown biomass in loblolly pine plantations in the Carolina flatwoods. **Forest Science** 28: 115-127.

Hereford, J.. 2001. Comparison of four parameter selection techniques. In **Proceedings of SoutheastCon 2001**. Clemson, SC, March 30 - April 1, 2001. p. 11 - 16.

Houghton, R. A.. 2002. Terrestrial carbon sinks- Uncertain explanations. **Biologist** 49: 155-160.

Houghton, R. A. and G. M. Woodwell. 1989. Global climatic change. **Scientific American** 260(4) : 36-44.

Hussin, Y. A., R. M. Reich, and R. M. Hoffer. 1991. Estimating slash pine biomass using radar backscatter. **IEEE Transactions on Geoscience and Remote Sensing** 29: 427-431.

Hyypä, J., H. Hyypä, M. Inkinen, M. Engdahl, S. Linko, Y.-H. Zhu. 2000. Accuracy comparison of various remote sensing data sources in the retrieval of forest stand attributes, **Forest Ecology and Management** 128: 109-120.

Hyypä, J. and M. Inkinen. 2002. Detecting and estimating attributes for single trees using laser scanner. **Photogrammetric Journal of Finland** 18: 43-53.

Imhoff, M. L.. 1995. Radar backscatter and biomass saturation: ramifications for global biomass inventory. **IEEE Transactions on Geoscience and Remote Sensing**, 33: 511-518.

Israelsson, H., J. Askne, and R. Sylvander, 1994. Potential of SAR for Forest Bole Volume Estimation. **International Journal of Remote Sensing** 15: 2809-2826

Jo, H. K. and E. G. McPherson. 1995. Carbon storage and flux in urban residential greenspace. **Journal of Environmental Management** 45: 109-133.

Johnson, H. B., H. W. Polley, and H. S. Mayeux. 1993. Increasing CO₂ and plant-plant interactions: effects on natural vegetation. **Vegetatio** 104-105: 157-170.

Johnson, A. F., P. M. Woodward, and S. J. Titus. 1989. Lodgepole pine and white spruce crown fuel weights predicted from height and crown width. **Canadian Journal of Forest Research** 19: 527-530.

Kuester, J. L. and J. H. Mize. 1973. **Optimization Techniques with Fortran**. New York: McGraw-Hill. 500p.

Landsberg, J. J. and R. H. Waring. 1997. A generalised model of forest productivity using simplified concepts of radiation-use efficiency, carbon balance and partitioning. **Forest Ecology and Management** 95: 209-228.

Larsen, M.. 1999. Finding an optimal match window for spruce top detection based on an optical tree model. In: D. A. Hill and D. G. Leckie (editors) **Automated Interpretation of High Spatial Resolution Digital Imagery for Forestry**. Victoria, British Columbia, Canada, February 10-12, 1998. Natural Resources Canada, Canadian Forest Service, Pacific Forestry Centre, Victoria, British Columbia. p. 55-66.

Lee, C. F., J. C. Lee, A. C. Lee. 2000. **Statistics for Business and Financial Economics**. Second edition. Singapore: World Scientific. 975p.

Lefsky, M. A., W. B. Cohen, S. A. Acker, G. G. Parker, T. A. Spies, and D. Harding. 1999. Lidar remote sensing of the canopy structure and biophysical properties of Douglas-fir western hemlock forests. **Remote Sensing of Environment** 70: 339-361.

Lefsky, M. A., W. B. Cohen, D. J. Harding, G. G. Parker, S. A. Acker, S. T. Gower. 2002. Lidar remote sensing of above-ground biomass in three biomes. **Global Ecology and Biogeography** 11: 393 - 399.

Lefsky, M. A., W. B. Cohen, and T. A. Spies. 2001. An evaluation of alternative remote sensing products for forest inventory, monitoring, and mapping of Douglas-fir forests in western Oregon. **Canadian Journal of Forest Research** 31: 78-87.

Lefsky, M. A., D. Harding, W. B. Cohen, G. Parker, and H. H. Shugart. 1999. Surface lidar remote sensing of basal area and biomass in deciduous forests of Eastern Maryland, USA. **Remote Sensing of Environment** 67: 83-98.

Le Toan, T., A. Beaudoin, J. Riou and D. Guyon. 1992. Relating forest biomass to SAR data, **IEEE Transactions on Geoscience and Remote Sensing** 30: 403-411.

Li, X., A. B. Baker, and T. Hutt. 2002. Accuracy of Airborne IFSAR **GISVision**, August, 2002. <http://www.gisvisionmag.com/200208/AccuracyIFSAR.pdf>.

Lillesand, T. M., R. W. Kiefer, J. W. Chipman. 2004. **Remote Sensing and Image Interpretation**. Fifth edition. New York: John Wiley and Sons. 763p.

Liu, J., J. M. Chen, J. Cihlar, and W. Chen. 1999. Net primary productivity distribution in the BOREAS region from a process model using satellite and surface data. **Journal of Geophysical Research** 104(D22): 27735-27754.

Lodhiyal, L. S. and N. Lodhiyal. 1997. Variation in biomass and net primary productivity in short rotation high density central Himalayan poplar plantations. **Forest Ecology and Management** 98: 167-179.

Malhi, Y., P. Meir, and S. Brown. 2002. Forests, carbon and global climate. **Philosophical Transactions of the Royal Society of London A** 360 (1797): 1567-1591.

Masters, T.. 1993. **Practical Neural Network Recipes in C++**. San Diego, CA: Academic Press. 493p.

McCombs, J. W., S. D. Roberts, and D. L. Evans. 2003. Influence of fusing lidar and multispectral imagery on remotely sensed estimates of stand density and mean tree height in a managed loblolly pine plantation. **Forest Science** 49: 457 - 466.

McMurtrie, R. E., Comins, H. N., Kirshbaum, M. U. F., Wang, Y. P.. 1992. Modifying existing forest growth models to take account of effects of elevated CO₂. **Australian Journal of Botany**. 40: 657-677.

McPherson, E. G.. 1998. Atmospheric carbon reduction by Sacramento's urban forest. **Journal of Arboriculture** 24: 215-223.

Means, J. E., S. A. Acker, D. J. Harding, J. B. Blair, M. A. Lefsky, W. B. Cohen, M. E. Harmon, and W. A. McKee. 1999. Use of large-footprint scanning airborne lidar to estimate forest stand characteristics in the Western Cascades of Oregon. **Remote Sensing of Environment** 67: 298-308.

Meeuwig, R. O., E. L. Miller, and J. D. Budy. 1979. Estimating pinyon and juniper fuel and biomass from aerial photographs. **Research Note** INT-274. Ogden, UT: USDA Forest Service Intermountain Forest and Range Experiment Station. 7p.

Meyer, H. A. and D. P. Worley. 1957. Volume determinations from aerial stand volume tables and their accuracy. **Journal of Forestry** 55: 368-372.

Naesset, E.. 1997. Estimating timber volume of forest stands using airborne laser scanner data. **Remote Sensing of Environment** 61: 246-253.

Naidu, S. L., E. H. DeLucia, and R. B. Thomas. 1998. Contrasting patterns of biomass allocation in dominant and suppressed loblolly pine. **Canadian Journal of Forest Research** 28: 1116 - 1124.

Nelder, J. A. and R. Mead. 1964. A simplex method for function minimization. **Computer Journal** 7: 308-313.

Nelson, R., W. Krabill, and J. Tonelli. 1988. Estimating forest biomass and volume using airborne laser data. **Remote Sensing of Environment** 24: 247 – 267.

Nemeth, J. C.. 1973. Dry matter production in young loblolly (*Pinus taeda L.*) and slash pine (*Pinus Elliotti Engelm.*) plantations. **Ecological Monographs** 43: 21-41.

Nilsson, A.. 1992. **Greenhouse Earth**. New York: Wiley. 219p.

Nowak, D.J.. 1994. Atmospheric carbon dioxide reduction by Chicago's urban forest. In: McPherson, E.G., Nowak, D.J., and Rowntree, R.A. (Eds.), Chicago's Urban Forest Ecosystem: Results of the Chicago Urban Forest Climate Project. **USDA Forest Service General Technical Report** NE-186, Radnor, PA. p. 83-94.

Nowak, D.J. and D.E. Crane. 2002. Carbon storage and sequestration by urban trees in the USA. **Environmental Pollution** 116: 381-389.

Ott, R. L.. 1993. **An Introduction to Statistical Methods and Data Analysis**. Belmont, CA: Duxbery Press. 1051p.

Phillips, D. L., S. L. Brown, P. E. Schroeder, and R. A. Birdsey. 2000. Toward error analysis of large-scale forest carbon budgets. **Global Ecology and Biogeography** 9: 305-313.

Pehl, C. E., C. L. Tuttle, J. N. Houser, and D. M. Moehring. 1984. Total biomass and nutrients of 25-year-old loblolly pines (*Pinus taeda L.*). **Forest Ecology and Management** 9: 155-160.

Pollock, R. J.. 1994. A model-based approach to automatically locating tree crowns in high spatial resolution images. In *Image and Signal Processing for Remote Sensing*. Rome, Italy. **Proceedings of SPIE** 2315. p. 526-537.

Popescu, S. C. and R. H. Wynne. 2002. Estimating plot-level tree heights with lidar: local filtering with a canopy-height based variable window size. **Computers and Electronics in Agriculture** 37: 71 - 95.

Popescu, S.C., R.H. Wynne and R.H. Nelson. 2003. Measuring individual tree crown diameter with lidar and assessing its influence on estimating forest volume and biomass. **Canadian Journal of Remote Sensing** 29: 564-577.

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. 1992. **Numerical Recipes in C**. Second edition. New York: Cambridge University Press. p. 408 - 412.

Proisy, C., E. Mougin, F. Fromard, V. Trichon, and K. A. Karam. 2002. On the influence of canopy structure on the radar backscattering of mangrove forests. **International Journal of Remote Sensing** 23: 4197 - 4210.

Pulliainen, J., M. Engdahl, and M. Hallikainen. 2003. Feasibility of multi-temporal interferometric SAR data for stand-level estimation of boreal forest stem volume. **Remote Sensing of Environment** 85: 397-409.

Pulliainen, J. T., Heiska, K., Hyypä, J. and Hallikainen, M. T., 1994, Backscattering properties of boreal forests at C- and X-band, **IEEE Transactions on Geoscience and Remote Sensing** 32: 1041-1050.

Radarsat International, 2002. Radarsat International Prices.
<http://www.rsi.ca/products/radarsat/rsat.pdf>.

Riley, R. H. and D. L. Phillips. 1997. Resolution and error in measuring land-cover change: Effects on estimating net carbon release from Mexican terrestrial ecosystems. **International Journal of Remote Sensing** 18: 121-137.

Roth, G. and M. D. Levine (1994) Geometric primitive extraction using a genetic algorithm. **IEEE Transactions on Pattern Analysis and Machine Intelligence**. 16:901-905

Running, S. W. and Coughlin, J. C.. 1988. A general model of forest ecosystem processes for regional applications. I. Hydrologic balance, canopy gas exchange and primary production processes. **Ecological Modeling**. 42: 125-154.

Running, S.W., and R.E. Hunt. 1993. Generalization of a forest ecosystem process model for other biomes, BIOME-BGC, and an application for global-scale models. In J.R. Ehleringer and C.B. Field (eds) **Scaling Physiologic Processes: Leaf to Globe**. San Diego, CA: Academic Press. p 141-158.

Shiver, B. D., and B. E. Borders. 1996. **Sampling Techniques for Forest Resource Inventory**. New York: John Wiley and Sons. 356p.

Shugart, H. H., L. Bourgeau-Chavez, and E. S. Kasischke. 2000. Determination of stand properties in boreal and temperate forests using high-resolution imagery. **Forest Science** 46: 478-486.

Soliday, 2000. Successful IKONOS launch offers new source of GIS data. **ArcNews**, Spring, 2000.

Space Imaging, 2001. Space Imaging announces new commercial pricing on IKONOS satellite imagery products. **News Releases**: December 13, 2001.

http://www.spaceimaging.com/newsroom/2001_newpricing.htm

Spot Image. 2002. **Spot Products and Services Price List**.

http://www.spotimage.fr/automne_modules_files/standard/public/p336_fileLINKEDFILE_price.pdf.

Spurr, S. H.. 1960. **Photogrammetry and Photo-Interpretation**. Second edition. New York: Ronald Press. 472p.

Steininger, M. K.. 2000. Satellite estimation of tropical secondary forest above-ground biomass: Data from Brazil and Bolivia. **International Journal of Remote Sensing** 21: 1139-1157.

Stiteler, W. M. and P. F. Hopkins. 2000. Using genetic algorithms to select tree crown templates for finding trees in digital imagery. **Proceedings of the ASPRS National Convention**, Washington, D. C., May 22-26, 2000.

Todd, K. W., F. Csillag, and P. M. Atkinson. 2003. Three-dimensional mapping of light transmittance and foliage distribution using lidar. **Canadian Journal of Remote Sensing** 29: 544 - 555.

Treitz, P.. 2001. Variogram analysis of high spatial resolution remote sensing data: An examination of boreal forest ecosystems. **International Journal of Remote Sensing** 22: 3895 - 3900.

U. S. Geological Survey. 2002. Enhanced Thematic Mapper Plus (ETM+). <http://edc.usgs.gov/products/satellite/landsat7.html>.

U. S. Forest Service. 2001. **Forest Inventory and Analysis**. <http://fia.fs.fed.us/about.htm>.

Van Lear, D. H., J. B. Wade, and M. J. Teuke. 1984. Biomass and nutrient content of a 41-year-old loblolly pine (*Pinus taeda L.*) Plantation on a poor site in South Carolina. **Forest Science** 30: 395-404.

Wehr, A., and U. Lohr. 1999. Airborne laser scanning- An introduction and overview. **ISPRS Journal of Photogrammetry and Remote Sensing** 54: 68-82.

Weiss, N., and M. Hasset. **Introductory Statistics**. Second edition. Reading, MA: Addison-Wesley. 672p.

Wright, A. H.. 1991. Genetic algorithms for real parameter optimization. In Rawlings, G. J. E. (ed.) **Foundations of Genetic Algorithms**. San Mateo, CA: Morgan Kaufmann. p. 205-218.

Wright, M. H.. 1996. Direct search methods: Once scorned, now respectable. In: D. F. Griffiths and G. A. Wetson (Eds.) **Numerical Analysis 1995: Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis**. Harlow, UK: Addison Wesley Longman. p. 191-208.

Wu, S. T.. 1987. Potential application of multipolarization SAR for pine-plantation biomass estimation. **IEEE Transactions on Geoscience and Remote Sensing** 25: 403-409.

Wuebbles, D. J. and J. Edmonds. 1991. **Primer on Greenhouse Gases**. Chelsea, MI: Lewis Publishers. 230p.

Wulder, M., K. O. Niemann and D. G. Goodenough (2000). Local maximum filtering for the extraction of tree locations and basal area from high spatial resolution imagery. **Remote Sensing of Environment** 73: 103-114.

Yu, S., S. De Backer, and H. Schroder. 2002. Genetic feature selection combined with composite fuzzy nearest neighbor classifiers for hyperspectral satellite imagery. **Pattern Recognition Letters** 23: 183-190.

Appendix A. The Nelder-Mead simplex based version of the program for setting parameters for the program that located and measures trees in orthophotographs

```
from Tkinter import *
import math
import random
import Image
import sys
import string
import genetic_parameters #A dummy function for passing variables
import images
import gc
import array

def ratio(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    x_dim, y_dim = in_image.size
    images.x_dim.append(x_dim)
    images.y_dim.append(y_dim)
    ratio_image = Image.new("F", (x_dim, y_dim), 0)
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            value_1, value_2, value_3 = in_image.getpixel((lcv_column, lcv_row))
            #ratio_value = float(value_2 - value_1)
            if genetic_parameters.ratio_method == "D":
                ratio_value = float(float(value_2) - float(value_1))
            if genetic_parameters.ratio_method == "C":
                ratio_value = float(float(value_1) - float(value_2))
            if genetic_parameters.ratio_method == "N":
                if value_1 + value_2 > 0:
                    ratio_value = float(value_1 - value_2) / float(value_1 + value_2)
                else:
                    ratio_value = 0
            if genetic_parameters.ratio_method == "I":
```

```

        ratio_value = value_1
    if genetic_parameters.ratio_method == "G":
        ratio_value = value_2
    ratio_image.putpixel((lcv_column, lcv_row), ratio_value)
images.ratio_images.append(ratio_image)

def edge(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    band_1, band_2, band_3 = in_image.split()
    if genetic_parameters.edge_method == "I":
        edge_raw = band_1
    else:
        edge_raw = band_2
    return edge_raw

def low_pass(source_image, image_number):
    lp_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    si_array = array.array("f")
    si_array.fromlist(list(source_image.getdata()))
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            if lcv_row == 0 or lcv_row == images.y_dim[image_number] - 1 or lcv_column == 0 or lcv_column ==
images.x_dim[image_number] - 1:
                value = si_array[lcv_row * x_dim + lcv_column]
                lp_image.putpixel((lcv_column, lcv_row), value)
            else:
                value = si_array[lcv_row * x_dim + lcv_column]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column - 1)]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column)]
                value = value + si_array[((lcv_row - 1) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column - 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column - 1))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column))]

```

```

value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column + 1))]
value = float(value)/float(9)
lp_image.putpixel((lcv_column, lcv_row), value)

```

```

return lp_image

```

```

def calculate_dog(SD):
    multipliers = []
    pi = math.pi
    sqrt2 = math.sqrt(2)
    for lcv_x in range(-5, 6):
        lcv_calculate = float(lcv_x)
        term_1 = -0.5 * sqrt2 / (math.sqrt(pi) * math.pow(SD, 3))
        term_2 = math.exp(-0.5 * ((lcv_calculate * lcv_calculate) / (SD * SD)))
        term_3 = 0.5 * math.sqrt(2) / (math.sqrt(pi) * math.pow(SD, 5))
        term_4 = (lcv_calculate) * (lcv_calculate)
        term_5 = math.exp(-0.5 * (lcv_calculate * lcv_calculate) / (SD * SD))
        coef = (term_1 * term_2) + (term_3 * term_4 * term_5)
        multipliers.append(coef)

```

```

return multipliers

```

```

def lmin(edge_raw, image_number, SD):
    lm_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    lmax_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    multipliers = calculate_dog(SD)
    #multipliers = [0, 1, 1, 1, 1, -8, 1, 1, 1, 0]
    edge_array = array.array("f")
    edge_array.fromlist(list(edge_raw.getdata()))

    pp_lm = lm_image.putpixel
    pp_lmax = lmax_image.putpixel

```

```

for lcv_row in range (6, images.y_dim[image_number] - 6):
    for lcv_column in range (6, images.x_dim[image_number] - 6):
        value_01 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[lcv_column-1 + (lcv_row + 1) * x_dim] +
edge_array[lcv_column+1 + (lcv_row - 1) * x_dim]
        value_1 = edge_array[(lcv_column-5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column-6) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row - 6) * x_dim]
        value_2 = edge_array[(lcv_column-4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column-5) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row - 5) * x_dim]
        value_3 = edge_array[(lcv_column-3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column-4) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row - 4) * x_dim]
        value_4 = edge_array[(lcv_column-2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column-3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row - 3) * x_dim]
        value_5 = edge_array[(lcv_column-1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column-2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row - 2) * x_dim]
        value_6 = edge_array[(lcv_column+1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
        value_7 = edge_array[(lcv_column+2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
        value_8 = edge_array[(lcv_column+3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 2) * x_dim]
        value_9 = edge_array[(lcv_column+4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 3) * x_dim]
        value_10 = edge_array[(lcv_column+5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row + 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row + 4) * x_dim]
        value_11 = edge_array[(lcv_column) + (lcv_row-5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 5) * x_dim]
        value_12 = edge_array[(lcv_column) + (lcv_row-4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 4) * x_dim]
        value_13 = edge_array[(lcv_column) + (lcv_row-3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 3) * x_dim]
        value_14 = edge_array[(lcv_column) + (lcv_row-2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 2) * x_dim]
        value_15 = edge_array[(lcv_column) + (lcv_row-1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 1) * x_dim]
        value_02 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row) * x_dim]

```

```

        value_16 = edge_array[(lcv_column) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
        value_17 = edge_array[(lcv_column) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 2) * x_dim]
        value_18 = edge_array[(lcv_column) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 3) * x_dim]
        value_19 = edge_array[(lcv_column) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 4) * x_dim]
        value_20 = edge_array[(lcv_column) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 5) * x_dim]
        value_21 = edge_array[(lcv_column-5) + (lcv_row) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 5 + (lcv_row + 1) * x_dim]
        value_22 = edge_array[(lcv_column-4) + (lcv_row) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 1) * x_dim]
        value_23 = edge_array[(lcv_column-3) + (lcv_row) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 1) * x_dim]
        value_24 = edge_array[(lcv_column-2) + (lcv_row) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 1) * x_dim]
        value_25 = edge_array[(lcv_column-1) + (lcv_row) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 1) * x_dim]
        value_03 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + (lcv_row + 1) * x_dim]
        value_26 = edge_array[(lcv_column+1) + (lcv_row) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
        value_27 = edge_array[(lcv_column+2) + (lcv_row) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row + 1) * x_dim]
        value_28 = edge_array[(lcv_column+3) + (lcv_row) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
        value_29 = edge_array[(lcv_column+4) + (lcv_row) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 1) * x_dim]
        value_30 = edge_array[(lcv_column+5) + (lcv_row) * x_dim] + edge_array[(lcv_column + 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 1) * x_dim]
        value_31 = edge_array[(lcv_column-5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 6) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 6) * x_dim]
        value_32 = edge_array[(lcv_column-4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 5) * x_dim]

```

```

        value_33 = edge_array[(lcv_column-3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 4) * x_dim]
        value_34 = edge_array[(lcv_column-2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 3) * x_dim]
        value_35 = edge_array[(lcv_column-1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row + 2) * x_dim]
        value_04 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
        value_36 = edge_array[(lcv_column+1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
        value_37 = edge_array[(lcv_column+2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row - 1) * x_dim]
        value_38 = edge_array[(lcv_column+3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row - 2) * x_dim]
        value_39 = edge_array[(lcv_column+4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row - 3) * x_dim]
        value_40 = edge_array[(lcv_column+5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row - 4) * x_dim]

```

```
lm_list = []
```

```

        lm_1 = multipliers[0]*value_1 + multipliers[1]*value_2 + multipliers[2]*value_3 + multipliers[3]*value_4 + multipliers[4] *
value_5 + multipliers[5] * value_01 + multipliers[6] * value_6 + multipliers[7]*value_7 + multipliers[8]*value_8 + multipliers[9] * value_9 + multipliers[10]
* value_10

```

```

        lm_2 = multipliers[0]*value_11 + multipliers[1]*value_12 + multipliers[2]*value_13 + multipliers[3]*value_14 + multipliers[4] *
value_15 + multipliers[5] * value_02 + multipliers[6] * value_16 + multipliers[7]*value_17 + multipliers[8]*value_18 + multipliers[9] * value_19 +
multipliers[10] * value_20

```

```

        lm_3 = multipliers[0]*value_21 + multipliers[1]*value_22 + multipliers[2]*value_23 + multipliers[3]*value_24 + multipliers[4] *
value_25 + multipliers[5] * value_03 + multipliers[6] * value_26 + multipliers[7]*value_27 + multipliers[8]*value_28 + multipliers[9] * value_29 +
multipliers[10] * value_30

```

```

        lm_4 = multipliers[0]*value_31 + multipliers[1]*value_32 + multipliers[2]*value_33 + multipliers[3]*value_34 + multipliers[4] *
value_35 + multipliers[5] * value_04 + multipliers[6] * value_36 + multipliers[7]*value_37 + multipliers[8]*value_38 + multipliers[9] * value_39 +
multipliers[10] * value_40

```

```
lm_list.append(lm_1)
```

```
lm_list.append(lm_2)
```

```
lm_list.append(lm_3)
```

```

        lm_list.append(lm_4)
    lm_list.sort()
    pp_lm((lcv_column, lcv_row), lm_list[3])
    pp_lmax((lcv_column, lcv_row), lm_list[0])
return lm_image, lmax_image

```

```

def make_int_maps(image_number):
    images.value_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    images.radius_image = Image.new("L", (images.x_dim[image_number], images.y_dim[image_number]), 0)

```

```

def find_value_diam(image_number, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay):

```

```

    if min_total < 0:
        min_total = 0

```

```

    edge_array = array.array("f")
    edge_array.fromlist(list(images.edge_images[image_number].getdata()))

```

```

    lmin_array = array.array("f")
    lmin_array.fromlist(list(images.lmin_image.getdata()))

```

```

    ratio_array = array.array("f")
    ratio_array.fromlist(list(images.ratio_images[image_number].getdata()))

```

```

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

```

```

    pp_value = images.value_image.putpixel
    pp_radius = images.radius_image.putpixel

```

```

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            test_1 = lmin_array[lcv_line * x_dim + lcv_column]
            test_2 = ratio_array[lcv_line * x_dim + lcv_column]
            if test_1 < center_thresh and test_2 > ratio_thresh:
                high_value = -999

```

```

high_radius = 0
stop = "N"
stop_now = "N"
for lcv_radius in range(genetic_parameters.min_radius, genetic_parameters.max_radius + 1):
    if stop == "Y":
        stop_now = "Y"
    pixels = 0
    value = 0
    sqrad = lcv_radius * lcv_radius
    for lcv_y in range(-1 * lcv_radius, lcv_radius + 1):
        x_radius = int(round(math.sqrt(sqrad - lcv_y * lcv_y)))
        position_y = lcv_line + lcv_y
        pos_position_x = lcv_column + x_radius
        neg_position_x = lcv_column - x_radius
        if ratio_array[position_y * x_dim + pos_position_x] < adj_thresh or edge_array[position_y * x_dim +
pos_position_x] < ae_thresh:
            stop = "Y"
        if ratio_array[position_y * x_dim + neg_position_x] < adj_thresh or edge_array[position_y * x_dim +
neg_position_x] < ae_thresh:
            stop = "Y"
        value = value + lmin_array[position_y * x_dim + pos_position_x]
        value = value + lmin_array[position_y * x_dim + neg_position_x]
        if x_radius == 0:
            pixels = pixels + 1
        else:
            pixels = pixels + 2

    total_value = float(value) / float(pixels)
    total_value = total_value + decay*lcv_radius
    if total_value > high_value:
        high_value = total_value
        high_radius = lcv_radius
    if high_value < min_total:
        high_value = 0
    if stop_now == "Y":
        break

```



```
pp_value((lcv_column, lcv_line), high_value)
pp_radius((lcv_column, lcv_line), high_radius)
```

```
def refine_trees(image_number, alpha, beta, gamma):
```

```
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
```

```
    lmax_array = array.array("f")
    lmax_array.fromlist(list(images.lmax_image.getdata()))
```

```
    radius_array = array.array("f")
    radius_array.fromlist(list(images.radius_image.getdata()))
```

```
    value_array = array.array("f")
    value_array.fromlist(list(images.value_image.getdata()))
```

```
    ratio_array = array.array("f")
    ratio_array.fromlist(list(images.ratio_images[image_number].getdata()))
```

```
    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
            if pixel_rad > 0:
                keep = 1
                score = alpha * value_array[lcv_line * x_dim + lcv_column]
                score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
                score = score + gamma * ratio_array[lcv_line * x_dim + lcv_column]
                for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                    for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                        radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                        ignore = "N"
                        if radius > pixel_rad:
                            ignore = "Y"
```

```

        if ignore == "N":
            wscore = alpha * value_array[win_y * x_dim + win_x]
            wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
            wscore = wscore + gamma * ratio_array[win_y * x_dim + win_x]
            if wscore > score:
                keep = 0
                break
        if int(keep) == 0:
            break
    images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

radius_array = array.array("f")
radius_array.fromlist(list(images.radius_image.getdata()))

#The following routine cleans up any small trees that are encroaching on a large tree's space

for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_array[lcv_line * x_dim + lcv_column]
            score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
            score = score + gamma * ratio_array[lcv_line * x_dim + lcv_column]
            for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                    radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                    ignore = "N"
                    if radius > pixel_rad:
                        ignore = "Y"
                    if ignore == "N" and radius_array[win_y * x_dim + win_x] > 0:
                        wscore = alpha * value_array[win_y * x_dim + win_x]
                        wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
                        wscore = wscore + gamma * ratio_array[win_y * x_dim + win_x]
                        if wscore < score:
                            images.radius_image.putpixel((win_x, win_y), 0)

```

```
images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)
```

```
def make_output(image_number):  
    trees = 0  
    for lcv_line in range (genetic_parameters.buffer, images.y_dim[image_number] - genetic_parameters.buffer):  
        for lcv_column in range (genetic_parameters.buffer, images.x_dim[image_number] - genetic_parameters.buffer):  
            pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))  
            center_x = float(images.x_dim[image_number]) / float(2)  
            center_y = float(images.y_dim[image_number]) / float(2)  
            term_1 = (center_x - float(lcv_column)) * (center_x - float(lcv_column))  
            term_2 = (center_y - float(lcv_line)) * (center_y - float(lcv_line))  
            distance = math.sqrt(term_1 + term_2)  
            if pixel_rad > 0 and int(round(distance)) <= genetic_parameters.plot_radius:  
                trees = trees + 1  
  
    return trees  
  
def start(image_number, alpha, beta, gamma, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay, SD):  
    genetic_parameters.counter = genetic_parameters.counter + 1  
    if genetic_parameters.first_run[image_number] == "Y":  
        ratio(image_number)  
    if genetic_parameters.first_run[image_number] == "Y":  
        images.lp_images.append(low_pass(images.ratio_images[image_number], image_number))  
    if genetic_parameters.first_run[image_number] == "Y":  
        if genetic_parameters.edge_method == "R":  
            images.lp_images.append(lp_image)  
        else:  
            edge_raw = edge(image_number)  
            images.edge_images.append(low_pass(edge_raw, image_number))  
    else:  
        edge_raw = edge(image_number)  
    images.lmin_image, images.lmax_image = lmin(edge_raw, image_number, SD)  
    make_int_maps(image_number)  
    find_value_diam(image_number, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay)
```

```

refine_trees(image_number, alpha, beta, gamma)
trees = make_output(image_number)
del images.radius_image
del images.value_image
return trees

def print_scores(best_score, best_score_index, parameter_matrix):

#
#This function prints out the scores, origin and parameters of the extant organisms
#
    parameter_names = ["Alpha:", "Beta:", "Gamma:", "Center thresh.:", "Ratio thresh.:", "A. R. thresh.:", "A. E. thresh.:", "Min. total:", "Decay:", "SD"]
    print "Information for organism: ", best_score_index
    print "Score: ", best_score
    print "Parameters:"
    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        print parameter_names[lcv_parameter], parameter_matrix[best_score_index][lcv_parameter]
    print "-----"

#
# Nelder and Meade Simplex Algorithm
#

def correl(array_1, array_2):
    N = len(array_1)
    if N <> len(array_2):
        print "CORREL error: Arrays must have the same number of elements"
        return 0
    sum_1 = 0.0
    sum_2 = 0.0
    for lcv_average in range (0, N):
        sum_1 = sum_1 + array_1[lcv_average]
        sum_2 = sum_2 + array_2[lcv_average]
    average_1 = sum_1 / float(N)
    average_2 = sum_2 / float(N)

```

```

sum_1 = 0.0
sum_2 = 0.0
cov_sum = 0.0
for lcv_sd in range (0, N):
    sum_1 = sum_1 + math.pow((array_1[lcv_sd] - average_1), 2)
    sum_2 = sum_2 + math.pow((array_2[lcv_sd] - average_2), 2)
    cov_sum = cov_sum + (array_1[lcv_sd] - average_1) * (array_2[lcv_sd] - average_2)
sd_1 = math.pow((sum_1 / float(N)), 0.5)
sd_2 = math.pow((sum_2 / float(N)), 0.5)
cov = cov_sum / float(N)

try:
    r = cov / (sd_1 * sd_2)
except:
    r = 0

return r

```

```

def eval_func (parameter_list):
    score = 0

```

```

    alpha = parameter_list[0]
    beta = parameter_list[1]
    gamma = parameter_list[2]
    center = parameter_list[3]
    ratio = parameter_list[4]
    adj = parameter_list[5]
    ae = parameter_list[6]
    min = parameter_list[7]
    decay = parameter_list[8]
    SD = parameter_list[9]

```

```

    predicted_array = []
    actual_array = []

```

```

    for image_number in range (0, genetic_parameters.number_test_images):

```

```

trees = start(image_number, alpha, beta, gamma, center, ratio, adj, ae, min, decay, SD)
genetic_parameters.first_run[image_number] = "N"
if image_number < genetic_parameters.number_correl_images:
    diff = trees - genetic_parameters.actual_number_trees[image_number]
    score = score + math.pow(diff, 2)
predicted_array.append(trees)
actual_array.append(genetic_parameters.actual_number_trees[image_number])

r = correl(predicted_array, actual_array)

if r > genetic_parameters.best_correl:
    print "Found better correlation. r =", r
    genetic_parameters.best_correl = r
    genetic_parameters.best_correl_parameters = parameter_list

return score

def create_initial_values(N, a, initial_parameter_list):

    sqrt2 = math.pow(2, 0.5)
    sqrtNp1 = math.pow(N+1, 0.5)

    parameter_matrix = []

    parameter_matrix.append(initial_parameter_list)

    for lcv_y in range (1, N+1):
        parameter_row = array.array("f", [])
        for lcv_x in range (0, N):
            p = (a[lcv_x] / (N * sqrt2)) * (sqrtNp1 + N - 1)
            q = (a[lcv_x] / (N * sqrt2)) * (sqrtNp1 - 1)
            if lcv_y == lcv_x + 1:
                value_to_add = initial_parameter_list[lcv_x] + p
            else:
                value_to_add = initial_parameter_list[lcv_x] + q
            parameter_row.append(value_to_add)

```

```

        parameter_matrix.append(parameter_row)

#score_list = array.array("f")
score_list = []

for lcv_y in range (0, N+1):
    score = eval_func(parameter_matrix[lcv_y])
    score_list.append(score)

return parameter_matrix, score_list

```

```
def calculate_reflected(N, parameter_matrix, score_list, alpha):
```

```

#
# Find the parameter value and number of the worst point
#

worst_score = score_list[0]
worst_point = 0

#
# Find the worst score and the number of the point with the worst score
#

for lcv in range (1, N+1):
    if score_list[lcv] > worst_score:
        worst_point = lcv
        worst_score = score_list[lcv]

#
# Find the centroids of all but the worst point
#

centroid_list = []

```

```

for lcv_x in range(0, N):
    parameter_sum = 0
    for lcv_y in range(0, N+1):
        parameter_sum = parameter_sum + parameter_matrix[lcv_y][lcv_x]
    centroid = (float(1) / float(N))* (parameter_sum - parameter_matrix[worst_point][lcv_x])
    centroid_list.append(centroid)

#
# Find the parameters for the reflected point
#

reflected_parameters = []
worst_list = []

for lcv in range (0, N):
    reflected_value = centroid_list[lcv] + alpha * (centroid_list[lcv] - parameter_matrix[worst_point][lcv])
    reflected_parameters.append(reflected_value)
    worst_list.append(parameter_matrix[worst_point][lcv])

return reflected_parameters, centroid_list, worst_list, worst_point, worst_score

def calculate_contracted(N, reflected_parameters, worst_parameters, centroid_list, was_it_the_worst, beta):

#
# Calculates the value of a contracted point
#

contracted_parameters = []

if was_it_the_worst == "yes":
    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - worst_parameters[lcv])
        contracted_parameters.append(new_parameter_value)
else:
    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - reflected_parameters[lcv])

```



```

        contracted_parameters.append(new_parameter_value)

    return contracted_parameters

def calculate_expansion(N, reflected_parameters, centroid_list, gamma):

    #
    # Calculates the value of an expansion point
    #

    expansion_parameters = []

    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] + gamma * (reflected_parameters[lcv] - centroid_list[lcv])
        expansion_parameters.append(new_parameter_value)

    return expansion_parameters

def move_incrementally(N, parameter_matrix, score_list):

    #
    # Moves the points in the parameter matrix incrementally closer to the parameters of the best member of the population
    #

    best_score = score_list[0]
    best_score_number = 0

    for lcv in range (1, N+1):
        if score_list[lcv] < best_score:
            best_score = score_list[lcv]
            best_score_number = lcv

    for lcv_x in range(0, N):
        for lcv_y in range(0, N+1):
            moved_point = (parameter_matrix[lcv_y][lcv_x] + parameter_matrix[best_score_number][lcv_x]) / float(2)

```

```

        parameter_matrix[lcv_y][lcv_x] = moved_point
score_list = array.array('f', [])
for lcv in range(0, N+1):
    score = eval_func(parameter_matrix[lcv])
    score_list.append(score)

return parameter_matrix, score_list

def find_best_score(N, score_list):
    best_score = score_list[0]
    best_score_index = 0
    for lcv in range(1, N+1):
        if score_list[lcv] < best_score:
            best_score = score_list[lcv]
            best_score_index = lcv

    return best_score, best_score_index

def check_for_stop(N, score_list):
    first_score = int(10 * score_list[0])
    stop = "yes"
    for lcv in range(1, N+1):
        if first_score <> int(10 * score_list[lcv]):
            stop = "no"

    return stop

def replace_point(N, worst_point, best_parameters, best_score, parameter_matrix, score_list):
    parameter_matrix[worst_point] = best_parameters
    score_list[worst_point] = best_score
    return parameter_matrix, score_list

def nelder_main():

    #
    # The main engine for the algorithm
    #

```

```

print "Up here..."

ofile = open(genetic_parameters.output_file, "w")

output_string = "Evaluations,Best score\n"

ofile.write(output_string)

images.ratio_images = []
images.lmin_images = []
images.lmax_images = []
images.x_dim = []
images.y_dim = []
images.lp_images = []
images.edge_images = []

genetic_parameters.first_run = []

for lcv in range (0, genetic_parameters.number_test_images):
    genetic_parameters.first_run.append("Y")

N = genetic_parameters.number_of_parameters
a = [5,5,5,15,25,30,70,10,30,1.5]
alpha = 1.0
beta = 0.5
gamma = 2.0
acc = 0.0001
initial_parameter_list = genetic_parameters.initial_parameter_list
max_iter = 1000
stop = "no"
current_iter = 0

parameter_matrix, score_list = create_initial_values(N, a, initial_parameter_list)

score_array = []

```

```

while (current_iter <= max_iter) and (stop == "no"):
    reflected_parameters, centroid_list, worst_list, worst_point, worst_score = calculate_reflected(N, parameter_matrix, score_list, alpha)
    reflected_score = eval_func(reflected_parameters)
    best_score, best_score_index = find_best_score(N, score_list)
    score_array.append(best_score)
    print "Best score = ", best_score
    print "Current iteration = ", current_iter
    #print "-----"

if reflected_score > worst_score:
    was_it_the_worst = "yes"
else:
    was_it_the_worst = "no"

if reflected_score < best_score:
    expansion_parameters = calculate_expansion(N, reflected_parameters, centroid_list, gamma)
    expansion_score = eval_func(expansion_parameters)
    if expansion_score < reflected_score:
        parameter_matrix, score_list = replace_point(N, worst_point, expansion_parameters, expansion_score, parameter_matrix,
score_list)

    else:
        parameter_matrix, score_list = replace_point(N, worst_point, reflected_parameters, reflected_score, parameter_matrix,
score_list)

    else:
        contraction_parameters = calculate_contracted(N, reflected_parameters, worst_list, centroid_list, was_it_the_worst, beta)
        contraction_score = eval_func(contraction_parameters)
        if contraction_score < worst_score:
            parameter_matrix, score_list = replace_point(N, worst_point, contraction_parameters, contraction_score, parameter_matrix,
score_list)

        else:
            parameter_matrix, score_list = move_incrementally(N, parameter_matrix, score_list)

current_iter = current_iter + 1

```

```

    stop = check_for_stop(N, score_list)
    if current_iter >= 100:
        if int(score_array[current_iter - 100]) == int(best_score):
            stop = "yes"
    best_score, best_score_index = find_best_score(N, score_list)
    return best_score, parameter_matrix[best_score_index]

```

```

def evaluate_scores(score, current_parameter_list, score_list, number_to_return):

```

```

    worst_score = score_list[0]
    worst_index = 0
    found_negative = "N"

    for lcv_eval in range(0, number_to_return):
        if (score_list[lcv_eval] < 0):
            worst_score = score_list[lcv_eval]
            worst_index = lcv_eval
            found_negative = "Y"
        elif (score_list[lcv_eval] > worst_score) and (found_negative == "N"):
            worst_score = score_list[lcv_eval]
            worst_index = lcv_eval

    kept = "no"

    if (worst_score > score) or (worst_score < 0):
        score_list[worst_index] = score
        genetic_parameters.starting_parameter_list[worst_index] = current_parameter_list
        kept = "yes"

    return score_list, kept

```

```

def find_starting_points (number_to_return, number_of_tries, min_list, max_list):

```

```

    ofile = open(genetic_parameters.output_file, "w")

```

```

output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"

ofile.write(output_string)

ofile.write(output_string)

genetic_parameters.first_run = []

for lcv in range(0, genetic_parameters.number_test_images):
    genetic_parameters.first_run.append("Y")

images.ratio_images = []
images.lmin_images = []
images.lmax_images = []
images.x_dim = []
images.y_dim = []
images.lp_images = []
images.edge_images = []

score_list = []

genetic_parameters.starting_parameter_list = []

genetic_parameters.verbose = "N"

for lcv_setup in range(0, number_to_return):
    parameters = []
    for lcv_param in range(0, genetic_parameters.number_of_parameters):
        parameters.append(1.1)
    score_list.append(-1)
    genetic_parameters.starting_parameter_list.append(parameters)

current_parameter_list = []

print "--> Generating starting points <--"

```

```

for lcv_try in range(0, number_of_tries):
    current_parameter_list = []
    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        value = min_list[lcv_parameter] + random.random() * (max_list[lcv_parameter] - min_list[lcv_parameter])
        current_parameter_list.append(value)
    score = eval_func(current_parameter_list)
    score_list, kept = evaluate_scores(score, current_parameter_list, score_list, number_to_return)
    print "Point", lcv_try, "Score =", score, "Kept =", kept
    if kept == "yes":
        print current_parameter_list

ofile.close()

```

```
def begin():
```

```

#
#This function will start the program out if it is being run as a stand alone program
#

genetic_parameters.best_correl = -1.0

genetic_parameters.best_correl_parameters = []

parameter_names = ["Alpha:", "Beta:", "Gamma:", "Center thresh.:", "Ratio thresh.:", "A. R. thresh.:", "A. E. thresh.:", "Min. total:", "Decay:", "SD"]

genetic_parameters.output_file = "c:/temp/pca_3_results.txt"

genetic_parameters.number_test_images = 25

genetic_parameters.number_correl_images = 25

image_names = []

genetic_parameters.image_names = []

```

```
genetic_parameters.image_names.append("1_test.tif")
genetic_parameters.image_names.append("2_test.tif")
genetic_parameters.image_names.append("3_test.tif")
genetic_parameters.image_names.append("4_test.tif")
genetic_parameters.image_names.append("5_test.tif")
genetic_parameters.image_names.append("6_test.tif")
genetic_parameters.image_names.append("7_test.tif")
genetic_parameters.image_names.append("8_test.tif")
genetic_parameters.image_names.append("9_test.tif")
genetic_parameters.image_names.append("10_test.tif")
genetic_parameters.image_names.append("11_test.tif")
genetic_parameters.image_names.append("12_test.tif")
genetic_parameters.image_names.append("13_test.tif")
genetic_parameters.image_names.append("14_test.tif")
genetic_parameters.image_names.append("15_test.tif")
genetic_parameters.image_names.append("16_test.tif")
genetic_parameters.image_names.append("17_test.tif")
genetic_parameters.image_names.append("18_test.tif")
genetic_parameters.image_names.append("19_test.tif")
genetic_parameters.image_names.append("20_test.tif")
genetic_parameters.image_names.append("21_test.tif")
genetic_parameters.image_names.append("22_test.tif")
genetic_parameters.image_names.append("23_test.tif")
genetic_parameters.image_names.append("24_test.tif")
genetic_parameters.image_names.append("25_test.tif")
```

```
genetic_parameters.actual_number_trees = []
```

```
genetic_parameters.actual_number_trees.append(103)
genetic_parameters.actual_number_trees.append(92)
genetic_parameters.actual_number_trees.append(119)
genetic_parameters.actual_number_trees.append(96)
genetic_parameters.actual_number_trees.append(90)
genetic_parameters.actual_number_trees.append(123)
genetic_parameters.actual_number_trees.append(104)
genetic_parameters.actual_number_trees.append(144)
```



```
genetic_parameters.actual_number_trees.append(106)
genetic_parameters.actual_number_trees.append(142)
genetic_parameters.actual_number_trees.append(138)
genetic_parameters.actual_number_trees.append(136)
genetic_parameters.actual_number_trees.append(136)
genetic_parameters.actual_number_trees.append(99)
genetic_parameters.actual_number_trees.append(160)
genetic_parameters.actual_number_trees.append(130)
genetic_parameters.actual_number_trees.append(121)
genetic_parameters.actual_number_trees.append(123)
genetic_parameters.actual_number_trees.append(136)
genetic_parameters.actual_number_trees.append(138)
genetic_parameters.actual_number_trees.append(99)
genetic_parameters.actual_number_trees.append(103)
genetic_parameters.actual_number_trees.append(105)
genetic_parameters.actual_number_trees.append(154)
genetic_parameters.actual_number_trees.append(110)
```

```
genetic_parameters.buffer = 10
```

```
genetic_parameters.plot_radius = 30
```

```
genetic_parameters.ratio_method = "C"
```

```
genetic_parameters.edge_method = "I"
```

```
genetic_parameters.filter = "L"
```

```
genetic_parameters.min_radius = 1
```

```
genetic_parameters.max_radius = 8
```

```
genetic_parameters.number_of_parameters = 10
```

```
genetic_parameters.counter = 0
```

```

min_list = [0, -5, 0, -5, -5, -10, 20, 0, -20, 1]
max_list = [5, 0, 5, 10, 20, 20, 90, 10, 10, 2.5]

number_to_return = 5

number_of_tries = 50

find_starting_points(number_to_return, number_of_tries, min_list, max_list)

for lcv_loop in range(0, number_to_return):

    print "-----"
    print "Trying starting point #", lcv_loop

    genetic_parameters.initial_parameter_list = genetic_parameters.starting_parameter_list[lcv_loop]

    print "Start parameters:", genetic_parameters.initial_parameter_list

final_score, final_parameter_list = nelder_main()

print "Best parameters for this iteration ="

for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
    print parameter_names[lcv_parameter], final_parameter_list[lcv_parameter]

print "Best correlation = ", genetic_parameters.best_correl

print "Best correlelation parameters = "

for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
    print parameter_names[lcv_parameter], genetic_parameters.best_correl_parameters[lcv_parameter]

print "Evaluations:", genetic_parameters.counter

print final_score

```

```

genetic_parameters.initial_parameter_list = final_parameter_list

converged = "no"

old_score = int(final_score)

while converged == "no":
    print "Attempting a restart at the convergence point"
    final_score, final_parameter_list = nelder_main()
    if lcv_loop == 0 or final_score < best_score:
        best_score = final_score
        best_parameter_list = final_parameter_list
    print "Final score = ", final_score
    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        print parameter_names[lcv_parameter], final_parameter_list[lcv_parameter]

    print "Best correlation = ", genetic_parameters.best_correl

    print "Best correlelation parameters = "

    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        print parameter_names[lcv_parameter], genetic_parameters.best_correl_parameters[lcv_parameter]

    if old_score == int(final_score):
        converged = "yes"
    else:
        old_score = int(final_score)
        genetic_parameters.initial_parameter_list = final_parameter_list

print "-----"
print "Best score = ", best_score
for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
    print parameter_names[lcv_parameter], best_parameter_list[lcv_parameter]
print "Evaluations:", genetic_parameters.counter

if __name__ == "__main__":
    begin()

```

Appendix B. The genetic algorithm based version of the program for setting parameters for the program that located and measures trees in orthophotographs

```
from Tkinter import *
import math
import random
import Image
import sys
import string
import genetic_parameters
import images
import gc
import array

def ratio(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    x_dim, y_dim = in_image.size
    images.x_dim.append(x_dim)
    images.y_dim.append(y_dim)
    ratio_image = Image.new("F", (x_dim, y_dim), 0)
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            value_1, value_2, value_3 = in_image.getpixel((lcv_column, lcv_row))
            #ratio_value = float(value_2 - value_1)
            if genetic_parameters.ratio_method == "D":
                ratio_value = float(float(value_2) - float(value_1))
            if genetic_parameters.ratio_method == "C":
                ratio_value = float(float(value_1) - float(value_2))
            if genetic_parameters.ratio_method == "N":
                if value_1 + value_2 > 0:
                    ratio_value = float(value_1 - value_2) / float(value_1 + value_2)
                else:
                    ratio_value = 0
```

```

        if genetic_parameters.ratio_method == "I":
            ratio_value = value_1
        if genetic_parameters.ratio_method == "G":
            ratio_value = value_2
        ratio_image.putpixel((lcv_column, lcv_row), ratio_value)
    images.ratio_images.append(ratio_image)

def edge(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    band_1, band_2, band_3 = in_image.split()
    if genetic_parameters.edge_method == "I":
        edge_raw = band_1
    else:
        edge_raw = band_2
    return edge_raw

def low_pass(source_image, image_number):
    lp_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    si_array = array.array("f")
    si_array.fromlist(list(source_image.getdata()))
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            if lcv_row == 0 or lcv_row == images.y_dim[image_number] - 1 or lcv_column == 0 or lcv_column ==
images.x_dim[image_number] - 1:
                value = si_array[lcv_row * x_dim + lcv_column]
                lp_image.putpixel((lcv_column, lcv_row), value)
            else:
                value = si_array[lcv_row * x_dim + lcv_column]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column - 1)]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column)]
                value = value + si_array[((lcv_row - 1) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column - 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column - 1))]

```

```

        value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column))]
        value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column + 1))]
        value = float(value)/float(9)
        lp_image.putpixel((lcv_column, lcv_row), value)

    return lp_image

def calculate_dog(SD):
    multipliers = []
    pi = math.pi
    sqrt2 = math.sqrt(2)
    for lcv_x in range(-5, 6):
        lcv_calculate = float(lcv_x)
        term_1 = -0.5 * sqrt2 / (math.sqrt(pi) * math.pow(SD, 3))
        term_2 = math.exp(-0.5 * ((lcv_calculate * lcv_calculate) / (SD * SD)))
        term_3 = 0.5 * math.sqrt(2) / (math.sqrt(pi) * math.pow(SD, 5))
        term_4 = (lcv_calculate) * (lcv_calculate)
        term_5 = math.exp(-0.5 * (lcv_calculate * lcv_calculate) / (SD * SD))
        coef = (term_1 * term_2) + (term_3 * term_4 * term_5)
        multipliers.append(coef)

    return multipliers

def lmin(edge_raw, image_number, SD):
    lm_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    lmax_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    multipliers = calculate_dog(SD)
    #multipliers = [0, 1, 1, 1, 1, -8, 1, 1, 1, 1, 0]
    edge_array = array.array("f")
    edge_array.fromlist(list(edge_raw.getdata()))

    pp_lm = lm_image.putpixel

```

```
pp_lmax = lmax_image.putpixel
```

```
for lcv_row in range (6, images.y_dim[image_number] - 6):
```

```
    for lcv_column in range (6, images.x_dim[image_number] - 6):
```

```
        value_01 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[lcv_column-1 + (lcv_row + 1) * x_dim] +  
edge_array[lcv_column+1 + (lcv_row - 1) * x_dim]  
        value_1 = edge_array[(lcv_column-5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column-6) + (lcv_row - 4) * x_dim] +  
edge_array[lcv_column - 4 + (lcv_row - 6) * x_dim]  
        value_2 = edge_array[(lcv_column-4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column-5) + (lcv_row - 3) * x_dim] +  
edge_array[lcv_column - 3 + (lcv_row - 5) * x_dim]  
        value_3 = edge_array[(lcv_column-3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column-4) + (lcv_row - 2) * x_dim] +  
edge_array[lcv_column - 2 + (lcv_row - 4) * x_dim]  
        value_4 = edge_array[(lcv_column-2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column-3) + (lcv_row - 1) * x_dim] +  
edge_array[lcv_column - 1 + (lcv_row - 3) * x_dim]  
        value_5 = edge_array[(lcv_column-1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column-2) + (lcv_row) * x_dim] +  
edge_array[lcv_column + (lcv_row - 2) * x_dim]  
        value_6 = edge_array[(lcv_column+1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column) + (lcv_row + 2) * x_dim] +  
edge_array[lcv_column + 2 + (lcv_row) * x_dim]  
        value_7 = edge_array[(lcv_column+2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row + 3) * x_dim] +  
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]  
        value_8 = edge_array[(lcv_column+3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row + 4) * x_dim] +  
edge_array[lcv_column + 4 + (lcv_row + 2) * x_dim]  
        value_9 = edge_array[(lcv_column+4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row + 5) * x_dim] +  
edge_array[lcv_column + 5 + (lcv_row + 3) * x_dim]  
        value_10 = edge_array[(lcv_column+5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row + 6) * x_dim] +  
edge_array[lcv_column + 6 + (lcv_row + 4) * x_dim]  
        value_11 = edge_array[(lcv_column) + (lcv_row-5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 5) * x_dim] +  
edge_array[lcv_column + 1 + (lcv_row - 5) * x_dim]  
        value_12 = edge_array[(lcv_column) + (lcv_row-4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 4) * x_dim] +  
edge_array[lcv_column + 1 + (lcv_row - 4) * x_dim]  
        value_13 = edge_array[(lcv_column) + (lcv_row-3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 3) * x_dim] +  
edge_array[lcv_column + 1 + (lcv_row - 3) * x_dim]  
        value_14 = edge_array[(lcv_column) + (lcv_row-2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 2) * x_dim] +  
edge_array[lcv_column + 1 + (lcv_row - 2) * x_dim]  
        value_15 = edge_array[(lcv_column) + (lcv_row-1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +  
edge_array[lcv_column + 1 + (lcv_row - 1) * x_dim]
```

```

value_02 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row) * x_dim]
value_16 = edge_array[(lcv_column) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
value_17 = edge_array[(lcv_column) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 2) * x_dim]
value_18 = edge_array[(lcv_column) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 3) * x_dim]
value_19 = edge_array[(lcv_column) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 4) * x_dim]
value_20 = edge_array[(lcv_column) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 5) * x_dim]
value_21 = edge_array[(lcv_column-5) + (lcv_row) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 5 + (lcv_row + 1) * x_dim]
value_22 = edge_array[(lcv_column-4) + (lcv_row) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 1) * x_dim]
value_23 = edge_array[(lcv_column-3) + (lcv_row) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 1) * x_dim]
value_24 = edge_array[(lcv_column-2) + (lcv_row) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 1) * x_dim]
value_25 = edge_array[(lcv_column-1) + (lcv_row) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 1) * x_dim]
value_03 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + (lcv_row + 1) * x_dim]
value_26 = edge_array[(lcv_column+1) + (lcv_row) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
value_27 = edge_array[(lcv_column+2) + (lcv_row) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row + 1) * x_dim]
value_28 = edge_array[(lcv_column+3) + (lcv_row) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
value_29 = edge_array[(lcv_column+4) + (lcv_row) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 1) * x_dim]
value_30 = edge_array[(lcv_column+5) + (lcv_row) * x_dim] + edge_array[(lcv_column + 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 1) * x_dim]
value_31 = edge_array[(lcv_column-5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 6) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 6) * x_dim]

```



```

        value_32 = edge_array[(lcv_column-4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 5) * x_dim]
        value_33 = edge_array[(lcv_column-3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 4) * x_dim]
        value_34 = edge_array[(lcv_column-2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 3) * x_dim]
        value_35 = edge_array[(lcv_column-1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row + 2) * x_dim]
        value_04 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
        value_36 = edge_array[(lcv_column+1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
        value_37 = edge_array[(lcv_column+2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row - 1) * x_dim]
        value_38 = edge_array[(lcv_column+3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row - 2) * x_dim]
        value_39 = edge_array[(lcv_column+4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row - 3) * x_dim]
        value_40 = edge_array[(lcv_column+5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row - 4) * x_dim]

```

```

lm_list = []

```

```

        lm_1 = multipliers[0]*value_1 + multipliers[1]*value_2 + multipliers[2]*value_3 + multipliers[3]*value_4 + multipliers[4] *
value_5 + multipliers[5] * value_01 + multipliers[6] * value_6 + multipliers[7]*value_7 + multipliers[8]*value_8 + multipliers[9] * value_9 + multipliers[10]
* value_10

```

```

        lm_2 = multipliers[0]*value_11 + multipliers[1]*value_12 + multipliers[2]*value_13 + multipliers[3]*value_14 + multipliers[4] *
value_15 + multipliers[5] * value_02 + multipliers[6] * value_16 + multipliers[7]*value_17 + multipliers[8]*value_18 + multipliers[9] * value_19 +
multipliers[10] * value_20

```

```

        lm_3 = multipliers[0]*value_21 + multipliers[1]*value_22 + multipliers[2]*value_23 + multipliers[3]*value_24 + multipliers[4] *
value_25 + multipliers[5] * value_03 + multipliers[6] * value_26 + multipliers[7]*value_27 + multipliers[8]*value_28 + multipliers[9] * value_29 +
multipliers[10] * value_30

```

```

        lm_4 = multipliers[0]*value_31 + multipliers[1]*value_32 + multipliers[2]*value_33 + multipliers[3]*value_34 + multipliers[4] *
value_35 + multipliers[5] * value_04 + multipliers[6] * value_36 + multipliers[7]*value_37 + multipliers[8]*value_38 + multipliers[9] * value_39 +
multipliers[10] * value_40

```

```

        lm_list.append(lm_1)

```

```

        lm_list.append(lm_2)
        lm_list.append(lm_3)
        lm_list.append(lm_4)
    lm_list.sort()
    pp_lm((lcv_column, lcv_row), lm_list[3])
    pp_lmax((lcv_column, lcv_row), lm_list[0])
return lm_image, lmax_image

```

```

def make_int_maps(image_number):
    images.value_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    images.radius_image = Image.new("L", (images.x_dim[image_number], images.y_dim[image_number]), 0)

```

```

def find_value_diam(image_number, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay):

```

```

    if min_total < 0:
        min_total = 0

```

```

    edge_array = array.array("f")
    edge_array.fromlist(list(images.edge_images[image_number].getdata()))

```

```

    lmin_array = array.array("f")
    lmin_array.fromlist(list(images.lmin_image.getdata()))

```

```

    ratio_array = array.array("f")
    ratio_array.fromlist(list(images.ratio_images[image_number].getdata()))

```

```

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

```

```

    pp_value = images.value_image.putpixel
    pp_radius = images.radius_image.putpixel

```

```

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            test_1 = lmin_array[lcv_line * x_dim + lcv_column]
            test_2 = ratio_array[lcv_line * x_dim + lcv_column]

```

```

if test_1 < center_thresh and test_2 > ratio_thresh:
    high_value = -999
    high_radius = 0
    stop = "N"
    stop_now = "N"
    for lcv_radius in range(genetic_parameters.min_radius, genetic_parameters.max_radius + 1):
        if stop == "Y":
            stop_now = "Y"
        pixels = 0
        value = 0
        sqrad = lcv_radius * lcv_radius
        for lcv_y in range (-1 * lcv_radius, lcv_radius + 1):
            x_radius = int(round(math.sqrt(sqrad - lcv_y * lcv_y)))
            position_y = lcv_line + lcv_y
            pos_position_x = lcv_column + x_radius
            neg_position_x = lcv_column - x_radius
            if ratio_array[position_y * x_dim + pos_position_x] < adj_thresh or edge_array[position_y * x_dim +
pos_position_x] < ae_thresh:
                stop = "Y"
            if ratio_array[position_y * x_dim + neg_position_x] < adj_thresh or edge_array[position_y * x_dim +
neg_position_x] < ae_thresh:
                stop = "Y"
            value = value + lmin_array[position_y * x_dim + pos_position_x]
            value = value + lmin_array[position_y * x_dim + neg_position_x]
            if x_radius == 0:
                pixels = pixels + 1
            else:
                pixels = pixels + 2

        total_value = float(value) / float(pixels)
        total_value = total_value + decay*lcv_radius
        if total_value > high_value:
            high_value = total_value
            high_radius = lcv_radius
        if high_value < min_total:
            high_value = 0

```

```

        if stop_now == "Y":
            break

        pp_value((lcv_column, lcv_line), high_value)
        pp_radius((lcv_column, lcv_line), high_radius)

def refine_trees(image_number, alpha, beta, gamma):

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

    lmax_array = array.array("f")
    lmax_array.fromlist(list(images.lmax_image.getdata()))

    radius_array = array.array("f")
    radius_array.fromlist(list(images.radius_image.getdata()))

    value_array = array.array("f")
    value_array.fromlist(list(images.value_image.getdata()))

    ratio_array = array.array("f")
    ratio_array.fromlist(list(images.ratio_images[image_number].getdata()))

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
            if pixel_rad > 0:
                keep = 1
                score = alpha * value_array[lcv_line * x_dim + lcv_column]
                score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
                score = score + gamma * ratio_array[lcv_line * x_dim + lcv_column]
                for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                    for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                        radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                        ignore = "N"

```

```

        if radius > pixel_rad:
            ignore = "Y"
        if ignore == "N":
            wscore = alpha * value_array[win_y * x_dim + win_x]
            wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
            wscore = wscore + gamma * ratio_array[win_y * x_dim + win_x]
            if wscore > score:
                keep = 0
                break
    if int(keep) == 0:
        break
    images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

```

```

radius_array = array.array("f")
radius_array.fromlist(list(images.radius_image.getdata()))

```

#The following routine cleans up any small trees that are encroaching on a large tree's space

```

for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_array[lcv_line * x_dim + lcv_column]
            score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
            score = score + gamma * ratio_array[lcv_line * x_dim + lcv_column]
            for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                    radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                    ignore = "N"
                    if radius > pixel_rad:
                        ignore = "Y"
                    if ignore == "N" and radius_array[win_y * x_dim + win_x] > 0:
                        wscore = alpha * value_array[win_y * x_dim + win_x]
                        wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
                        wscore = wscore + gamma * ratio_array[win_y * x_dim + win_x]

```

```

    if wscore < score:
        images.radius_image.putpixel((win_x, win_y), 0)

```

```

def refine_trees_old(image_number, alpha, beta, gamma, examination_fraction):
    if examination_fraction > 1:
        examination_fraction = float(1)
    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))
            if pixel_rad > 0:
                keep = 1
                score = alpha * images.value_image.getpixel((lcv_column, lcv_line))
                score = score + beta * images.lmin_image.getpixel((lcv_column, lcv_line))
                score = score + gamma * images.ratio_images[image_number].getpixel((lcv_column, lcv_line))
                for win_x in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                    for win_y in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                        radius = math.sqrt((win_x - lcv_line) * (win_x - lcv_line) + (win_y - lcv_column) * (win_y - lcv_column))
                        ignore = "N"
                        if radius > pixel_rad * examination_fraction:
                            ignore = "Y"
                        if ignore == "N":
                            wscore = alpha * images.value_image.getpixel((win_y, win_x))
                            wscore = wscore + beta * images.lmin_image.getpixel((win_y, win_x))
                            wscore = wscore + gamma * images.ratio_images[image_number].getpixel((win_y, win_x))
                            if wscore > score:
                                keep = 0
                                break
                    if int(keep) == 0:
                        break
                images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

```

```

def make_output(image_number):

```

```

trees = 0
for lcv_line in range (genetic_parameters.buffer, images.y_dim[image_number] - genetic_parameters.buffer):
    for lcv_column in range (genetic_parameters.buffer, images.x_dim[image_number] - genetic_parameters.buffer):
        pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))
        center_x = float(images.x_dim[image_number]) / float(2)
        center_y = float(images.y_dim[image_number]) / float(2)
        term_1 = (center_x - float(lcv_column)) * (center_x - float(lcv_column))
        term_2 = (center_y - float(lcv_line)) * (center_y - float(lcv_line))
        distance = math.sqrt(term_1 + term_2)
        if pixel_rad > 0 and int(round(distance)) <= genetic_parameters.plot_radius:
            trees = trees + 1

return trees

def start(image_number, first_run, alpha, beta, gamma, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay, SD):

    if first_run[image_number] == "Y":
        ratio(image_number)
    if first_run[image_number] == "Y":
        images.lp_images.append(low_pass(images.ratio_images[image_number], image_number))
    if first_run[image_number] == "Y":
        if genetic_parameters.edge_method == "R":
            images.lp_images.append(lp_image)
        else:
            edge_raw = edge(image_number)
            images.edge_images.append(low_pass(edge_raw, image_number))
    else:
        edge_raw = edge(image_number)
    images.lmin_image, images.lmax_image = lmin(edge_raw, image_number, SD)
    make_int_maps(image_number)
    find_value_diam(image_number, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay)
    refine_trees(image_number, alpha, beta, gamma)
    trees = make_output(image_number)
    del images.radius_image
    del images.value_image
    return trees

```

```

def check_duplicate(origin_selected, parameters_1, parameters_2, number_of_parameters):

#
#Checks to make sure that the two mates are not identical. If they are, a mutation, not a crossover, takes place.
#

    origin = "mutation"

    for lcv_parameter in range (0, number_of_parameters):
        int_parm_1 = long(parameters_1[lcv_parameter] * 10000)
        int_parm_2 = long(parameters_2[lcv_parameter] * 10000)
        if int_parm_1 != int_parm_2:
            origin = origin_selected

    return origin

def roulette (score_list, population_size):

#
#This function uses the roulette procedure to determine which organisms will breed
#

#
#Step 1: Create a new array containing the cumulative fitnesses
#

    cumulative = []
    cumulative.append(float(1) / math.sqrt(float(score_list[0] + 1)))

    for lcv_cumulative in range (1, population_size):
        cumulative.append(cumulative[lcv_cumulative - 1] + float(1) / math.sqrt(float(score_list[lcv_cumulative] + 1)))

#
#Step 2: Choose the lucky couple
#

```



```

random_1 = random.random() * cumulative[population_size - 1]
random_2 = random.random() * cumulative[population_size - 1]

found_1 = 0
found_2 = 0

mate_1 = 0
mate_2 = 0

for lcv_luck in range(0, population_size):
    if found_1 == 0 and random_1 > cumulative[lcv_luck]:
        mate_1 = lcv_luck
        found_1 = 1
    if found_2 == 0 and random_2 > cumulative[lcv_luck]:
        mate_2 = lcv_luck
        found_2 = 1

if mate_1 == mate_2:
    while 1:
        random_2 = random.random() * cumulative[population_size - 1]
        for lcv_luck in range(0, population_size):
            if random_2 > cumulative[lcv_luck]:
                mate_2 = lcv_luck
        if mate_1 == mate_2:
            pass
        else:
            break

#
#Return mates
#

return mate_1, mate_2

```

```

def crossover(parameters_mate_1, parameters_mate_2, number_of_parameters):

```

```

#
#The function performs a uniform crossover
#
child_parameters = []

for lcv_parameter in range(0, number_of_parameters):
    if random.random() < 0.5:
        child_parameters.append(parameters_mate_1[lcv_parameter])
    else:
        child_parameters.append(parameters_mate_2[lcv_parameter])

return child_parameters

def across(parameters_mate_1, parameters_mate_2, number_of_parameters):
#
#This function performs an average crossover
#

child_parameters = []

for lcv_parameter in range(0, number_of_parameters):
    parameter_value = (parameters_mate_1[lcv_parameter] + parameters_mate_2[lcv_parameter]) / 2
    child_parameters.append(parameter_value)

return child_parameters

def mutation(parameters_mate_1, parameters_mate_2, parameter_min, parameter_max, mutation_strength, number_of_parameters):
#
#This function performs mutations on a randomly chosen mate
#

if random.random() < 0.5:
    mate = 1

```

```

else:
    mate = 2
    child_parameters = []

    for lcv_parameter in range(0, number_of_parameters):
        parameter_to_mutate = int(random.random() * number_of_parameters)
        if lcv_parameter == parameter_to_mutate:
            amount_to_mutate = random.gauss(0, (parameter_max[parameter_to_mutate] - parameter_min[parameter_to_mutate]) *
mutation_strength)
            if mate == 1:
                child_parameters.append(parameters_mate_1[lcv_parameter] + amount_to_mutate)
            else:
                child_parameters.append(parameters_mate_2[lcv_parameter] + amount_to_mutate)
        else:
            if mate == 1:
                child_parameters.append(parameters_mate_1[lcv_parameter])
            else:
                child_parameters.append(parameters_mate_2[lcv_parameter])

    return child_parameters

def first_generation(parameter_min, parameter_max, number_of_parameters, population_size):
    #
    #This function creates the first generation of organisms
    #

    population_parameters = []

    for lcv_population in range (0, population_size):
        parameter_list = []
        for lcv_parameter in range (0, number_of_parameters):
            parameter_value = float(parameter_min[lcv_parameter]) + (float(parameter_max[lcv_parameter]) -
float(parameter_min[lcv_parameter])) * random.random()
            parameter_list.append(parameter_value)
        population_parameters.append(parameter_list)

```

```

    return population_parameters

def random_origin(population_size):

#
#This function is called prior to the first breeding. It assigns a random origin (crossover or mutation) to each organism.
#

    origin = []

    for lcv_population in range(0, population_size):
        random_number = random.random()
        if random_number < 0.33:
            origin.append("crossover")
        if random_number >= 0.33 and random_number < 0.67:
            origin.append("avg_crossover")
        if random_number >= 0.67:
            origin.append("mutation")

    return origin

def determine_origin(origin, population_size):

#
#Determines the origin of the next population member
#

    count_crossover = 0
    count_across = 0
    count_mutation = 0

    for lcv_population in range(0, population_size):
        if origin[lcv_population] == "crossover":
            count_crossover = count_crossover + 1
        if origin[lcv_population] == "avg_crossover":

```

```

        count_across = count_across + 1
    if origin[lcv_population] == "mutation":
        count_mutation = count_mutation + 1

    fraction_crossover = float(count_crossover) / float(population_size)
    fraction_across = float(count_across) / float(population_size)
    fraction_mutation = float(count_mutation) / float(population_size)

#
#The following lines ensure that a reproduction mode doesn't become extinct
#

shift = 0
shift_count = 0
crossover_shift = "N"
across_shift = "N"
mutation_shift = "N"

if fraction_crossover < 0.1:
    shift = 0.1 - fraction_crossover
    fraction_crossover = 0.1
    shift_count = 1
    crossover_shift = "Y"

if fraction_across < 0.1:
    large_shift = shift + 0.1 - fraction_across
    fraction_across = 0.1
    shift_count = shift_count + 1
    across_shift = "Y"

if fraction_mutation < 0.1:
    large_shift = shift + 0.1 - fraction_across
    fraction_mutation = 0.1
    shift_count = shift_count + 1
    mutation_shift = "Y"

```

```

if shift_count > 0:
    if crossover_shift == "N":
        fraction_crossover = fraction_crossover - shift / float(shift_count)
    if across_shift == "N":
        fraction_across = fraction_across - shift / float(shift_count)
    if mutation_shift == "N":
        fraction_mutation = fraction_mutation - shift / float(shift_count)

#
#These lines choose the reproduction mode
#

random_number = random.random()

if random_number < fraction_crossover:
    origin_selected = "crossover"
if random_number >= fraction_crossover and random_number < fraction_crossover + fraction_across:
    origin_selected = "avg_crossover"
if random_number >= fraction_crossover + fraction_across:
    origin_selected = "mutation"

return origin_selected, fraction_crossover, fraction_across, fraction_mutation

def compute_score(actual_number_trees, number_trees_found, number_test_images):

#
#This function computes the score associated with each test
#

score = 0

for lcv_test in range(0, number_test_images):
    difference = actual_number_trees[lcv_test] - float(number_trees_found[lcv_test])
    score = score + math.pow(difference, 2)

return score

```

```
def replace_values(weakest_member, score, score_list, child_parameters, population_parameters, origin_selected, origin, population_size):
```

```
#  
#This function replaces the score and parameters of an organism with those of a stronger child. Needless  
# to say it's pretty annoying that Python can't do this internally!  
#
```

```
    new_score_array = []  
    new_parameter_array = []  
    new_origin_array = []
```

```
    for lcv_population in range(0, population_size):  
        if lcv_population == weakest_member:  
            new_score_array.append(score)  
            new_parameter_array.append(child_parameters)  
            new_origin_array.append(origin_selected)  
  
        else:  
            new_score_array.append(score_list[lcv_population])  
            new_parameter_array.append(population_parameters[lcv_population])  
            new_origin_array.append(origin[lcv_population])
```

```
    return new_score_array, new_parameter_array, new_origin_array
```

```
def survivors(score, score_list, population_parameters, child_parameters, origin_selected, origin, population_size):
```

```
#  
#This function sees if the child is fitter than any of the existing members of the polulation.  
# If it is, it replaces the weakest member.  
#
```

```
    weaker_found = "N"  
    weakest_member = 0  
    weakest_score = 0
```

```

for lcv_population in range (0, population_size):
    if weaker_found == "Y" and score < score_list[lcv_population] and weakest_score < score_list[lcv_population]:
        weakest_score = score_list[lcv_population]
        weakest_member = lcv_population
    if weaker_found == "N" and score < score_list[lcv_population]:
        weaker_found = "Y"
        weakest_score = score_list[lcv_population]
        weakest_member = lcv_population

if weaker_found == "Y":
    score_list, population_parameters, origin = replace_values(weakest_member, score, score_list, child_parameters, population_parameters,
origin_selected, origin, population_size)

min_score = score_list[0]

for lcv_population in range (1, population_size):
    if score_list[lcv_population] < min_score:
        min_score = score_list[lcv_population]

return score_list, population_parameters, origin, min_score

def binder(alpha, beta, gamma, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay, SD):
#
#This function is used to bind the upper and lower parameter limits into arrays
#

new_array = []

new_array.append(alpha)
new_array.append(beta)
new_array.append(gamma)
new_array.append(center_thresh)
new_array.append(ratio_thresh)
new_array.append(adj_thresh)
new_array.append(ae_thresh)

```



```

    new_array.append(min_total)
    new_array.append(decay)
    new_array.append(SD)

    return new_array

def print_scores(score_list, population_parameters, origin, population_size, number_of_parameters, iteration):

#
#This function prints out the scores, origin and parameters of the extant organisms
#
    if iteration == 0 or ((iteration + 1) % 100) == 0:
        parameter_names = ["Alpha:", "Beta:", "Gamma:", "Center thresh.:", "Ratio thresh.:", "A. R. thresh.:", "A. E. thresh.:", "Min. total:",
"Decay:", "SD"]
        lowest_score = score_list[0]
        lowest_member = 0
        for lcv_population in range(1, population_size):
            if score_list[lcv_population] < lowest_score:
                lowest_score = score_list[lcv_population]
                lowest_member = lcv_population
        print "-----"
        print "Iteration:", iteration + 1
        print "Score = ", lowest_score
        print "Origin = ", origin[lowest_member]
        print "Parameters:"
        for lcv_parameter in range(0, number_of_parameters):
            print parameter_names[lcv_parameter], population_parameters[lowest_member][lcv_parameter]
        print "-----"

def main_function():

#
#This function drives the genetic algorithm.
#

    ofile = open(genetic_parameters.output_file, "w")

```

```

output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"

ofile.write(output_string)

ofile.write(output_string)

first_run = []
images.ratio_images = []
images.lmin_images = []
images.lmax_images = []
images.x_dim = []
images.y_dim = []
images.lp_images = []
images.edge_images = []

for lcv_first_run in range(0, genetic_parameters.number_test_images):
    first_run.append("Y")

#
#This section of the function initializes the population and computes the scores for the original members
#

population_parameters = first_generation(genetic_parameters.parameter_min, genetic_parameters.parameter_max,
    genetic_parameters.number_of_parameters, genetic_parameters.population_size)

origin = random_origin(genetic_parameters.population_size)

score_list = []

print "Initializing the original population."

for lcv_population in range(0, genetic_parameters.population_size):
    print "\nAnalyzing organism ", lcv_population
    alpha = population_parameters[lcv_population][0]
    beta = population_parameters[lcv_population][1]
    gamma = population_parameters[lcv_population][2]

```

```

center_thresh = population_parameters[lcv_population][3]
ratio_thresh = population_parameters[lcv_population][4]
adj_thresh = population_parameters[lcv_population][5]
ae_thresh = population_parameters[lcv_population][6]
min_total = population_parameters[lcv_population][7]
decay = population_parameters[lcv_population][8]
SD = population_parameters[lcv_population][9]

number_trees_found = []

for lcv_image in range(0, genetic_parameters.number_test_images):
    trees = start(lcv_image, first_run, alpha, beta, gamma, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay, SD)
    if first_run[lcv_image] == "Y":
        first_run[lcv_image] = "N"
    number_trees_found.append(trees)
    output_string = "In image " + str(lcv_image) + ", " + str(trees) + " trees were found."
    print output_string

score = compute_score(genetic_parameters.actual_number_trees, number_trees_found, genetic_parameters.number_test_images)

print "The score is: ", score

score_list.append(score)

#
#This section performs the genetic selection
#

print "\nPerforming genetic selection\n"

for lcv_iteration in range(0, genetic_parameters.iterations):
    mate_1, mate_2 = roulette(score_list, genetic_parameters.population_size)
    origin_selected, probab_crossover, probab_across, probab_mutation = determine_origin(origin, genetic_parameters.population_size)
    if origin_selected == "crossover" or origin_selected == "avg_crossover":

```

```

        origin_selected = check_duplicate(origin_selected, population_parameters[mate_1], population_parameters[mate_2],
genetic_parameters.number_of_parameters)
        if origin_selected == "crossover":
            child_parameters = crossover(population_parameters[mate_1], population_parameters[mate_2],
genetic_parameters.number_of_parameters)
        if origin_selected == "avg_crossover":
            child_parameters = across(population_parameters[mate_1], population_parameters[mate_2],
genetic_parameters.number_of_parameters)
        if origin_selected == "mutation":
            child_parameters = mutation(population_parameters[mate_1], population_parameters[mate_2], genetic_parameters.parameter_min,
genetic_parameters.parameter_max, genetic_parameters.mutation_strength, genetic_parameters.number_of_parameters)
    #print origin_selected
    alpha = child_parameters[0]
    beta = child_parameters[1]
    gamma = child_parameters[2]
    center_thresh = child_parameters[3]
    ratio_thresh = child_parameters[4]
    adj_thresh = child_parameters[5]
    ae_thresh = child_parameters[6]
    min_total = child_parameters[7]
    decay = child_parameters[8]
    SD = child_parameters[9]

    number_trees_found = []

    for lcv_image in range (0, genetic_parameters.number_test_images):
        trees = start(lcv_image, first_run, alpha, beta, gamma, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay, SD)
        number_trees_found.append(trees)
        #output_string = "In image " + str(lcv_image) + ", " + str(trees) + " trees were found."
        #print output_string

    score = compute_score(genetic_parameters.actual_number_trees, number_trees_found, genetic_parameters.number_test_images)

    #print "Score = ", score

    score_list, population_parameters, origin, best_score = survivors(score, score_list, population_parameters, child_parameters, origin_selected,

```

```

origin, genetic_parameters.population_size)

        print_scores(score_list, population_parameters, origin, genetic_parameters.population_size, genetic_parameters.number_of_parameters,
lcv_iteration)

        output_string = str(lcv_iteration) + "," + str(best_score) + "," + str(prob_crossover) + "," + str(prob_across) + "," + str(prob_mutation) + "\n"
        ofile.write(output_string)
    ofile.close()

def begin():

#
#This function will start the program out if it is being run as a stand alone program
#

    genetic_parameters.output_file = "genetic_engine_bake.csv"

    genetic_parameters.alpha_min = float(0)
    genetic_parameters.alpha_max = float(5)
    genetic_parameters.beta_min = float(-5)
    genetic_parameters.beta_max = float(0)
    genetic_parameters.gamma_min = float(0)
    genetic_parameters.gamma_max = float(5)
    genetic_parameters.center_thresh_min = float(-5)
    genetic_parameters.center_thresh_max = float(10)
    genetic_parameters.ratio_thresh_min = float(-5)
    genetic_parameters.ratio_thresh_max = float(20)
    genetic_parameters.adj_thresh_min = float(-10)
    genetic_parameters.adj_thresh_max = float(20)
    genetic_parameters.ae_thresh_min = float(20)
    genetic_parameters.ae_thresh_max = float(90)
    genetic_parameters.min_total_min = float(0)
    genetic_parameters.min_total_max = float(10)
    genetic_parameters.decay_min = float(-20)
    genetic_parameters.decay_max = float(10)
    genetic_parameters.SD_min = float(1)

```

```
genetic_parameters.SD_max = float(2.5)

genetic_parameters.parameter_min = binder(genetic_parameters.alpha_min,
    genetic_parameters.beta_min, genetic_parameters.gamma_min, genetic_parameters.center_thresh_min,
    genetic_parameters.ratio_thresh_min, genetic_parameters.adj_thresh_min, genetic_parameters.ae_thresh_min,
    genetic_parameters.min_total_min, genetic_parameters.decay_min, genetic_parameters.SD_min)

genetic_parameters.parameter_max = binder(genetic_parameters.alpha_max,
    genetic_parameters.beta_max, genetic_parameters.gamma_max, genetic_parameters.center_thresh_max,
    genetic_parameters.ratio_thresh_max, genetic_parameters.adj_thresh_max, genetic_parameters.ae_thresh_max,
    genetic_parameters.min_total_max, genetic_parameters.decay_max, genetic_parameters.SD_max)

genetic_parameters.plot_radius = 30

genetic_parameters.iterations = 5000

genetic_parameters.population_size = 100

genetic_parameters.mutation_strength = 0.3

genetic_parameters.number_test_images = 25

genetic_parameters.image_names = []

genetic_parameters.image_names.append("1_test.tif")
genetic_parameters.image_names.append("2_test.tif")
genetic_parameters.image_names.append("3_test.tif")
genetic_parameters.image_names.append("4_test.tif")
genetic_parameters.image_names.append("5_test.tif")
genetic_parameters.image_names.append("6_test.tif")
genetic_parameters.image_names.append("7_test.tif")
genetic_parameters.image_names.append("8_test.tif")
genetic_parameters.image_names.append("9_test.tif")
genetic_parameters.image_names.append("10_test.tif")
genetic_parameters.image_names.append("11_test.tif")
genetic_parameters.image_names.append("12_test.tif")
```

```
genetic_parameters.image_names.append("13_test.tif")
genetic_parameters.image_names.append("14_test.tif")
genetic_parameters.image_names.append("15_test.tif")
genetic_parameters.image_names.append("16_test.tif")
genetic_parameters.image_names.append("17_test.tif")
genetic_parameters.image_names.append("18_test.tif")
genetic_parameters.image_names.append("19_test.tif")
genetic_parameters.image_names.append("20_test.tif")
genetic_parameters.image_names.append("21_test.tif")
genetic_parameters.image_names.append("22_test.tif")
genetic_parameters.image_names.append("23_test.tif")
genetic_parameters.image_names.append("24_test.tif")
genetic_parameters.image_names.append("25_test.tif")
genetic_parameters.actual_number_trees = []
```

```
genetic_parameters.actual_number_trees.append(103)
genetic_parameters.actual_number_trees.append(92)
genetic_parameters.actual_number_trees.append(119)
genetic_parameters.actual_number_trees.append(96)
genetic_parameters.actual_number_trees.append(90)
genetic_parameters.actual_number_trees.append(123)
genetic_parameters.actual_number_trees.append(104)
genetic_parameters.actual_number_trees.append(144)
genetic_parameters.actual_number_trees.append(106)
genetic_parameters.actual_number_trees.append(142)
genetic_parameters.actual_number_trees.append(138)
genetic_parameters.actual_number_trees.append(136)
genetic_parameters.actual_number_trees.append(136)
genetic_parameters.actual_number_trees.append(99)
genetic_parameters.actual_number_trees.append(160)
genetic_parameters.actual_number_trees.append(130)
genetic_parameters.actual_number_trees.append(121)
genetic_parameters.actual_number_trees.append(123)
genetic_parameters.actual_number_trees.append(136)
genetic_parameters.actual_number_trees.append(138)
genetic_parameters.actual_number_trees.append(99)
```

```
genetic_parameters.actual_number_trees.append(103)
genetic_parameters.actual_number_trees.append(105)
genetic_parameters.actual_number_trees.append(154)
genetic_parameters.actual_number_trees.append(110)

genetic_parameters.buffer = 10

genetic_parameters.ratio_method = "C"

genetic_parameters.edge_method = "I"

genetic_parameters.filter = "L"

genetic_parameters.min_radius = 1

genetic_parameters.max_radius = 8

genetic_parameters.number_of_parameters = 10

main_function()

if __name__ == "__main__":
    begin()
```


Appendix C. A program for using the parameters found by the program in Appendix I or II to determine the number of trees in test images.

```
from Tkinter import *
import math
import random
import Image
import sys
import string
import genetic_parameters
import images
import gc
import array

def ratio(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    x_dim, y_dim = in_image.size
    images.x_dim.append(x_dim)
    images.y_dim.append(y_dim)
    ratio_image = Image.new("F", (x_dim, y_dim), 0)
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            value_1, value_2, value_3 = in_image.getpixel((lcv_column, lcv_row))
            if genetic_parameters.ratio_method == "D":
                ratio_value = float(float(value_2) - float(value_1))
            if genetic_parameters.ratio_method == "C":
                ratio_value = float(float(value_1) - float(value_2))
            if genetic_parameters.ratio_method == "N":
                if value_1 + value_2 > 0:
                    ratio_value = float(value_1 - value_2) / float(value_1 + value_2)
                else:
                    ratio_value = 0
            if genetic_parameters.ratio_method == "I":
                ratio_value = value_1
            if genetic_parameters.ratio_method == "G":
                ratio_value = value_2
```

```

        ratio_image.putpixel((lcv_column, lcv_row), ratio_value)
    images.ratio_images.append(ratio_image)

def edge(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    band_1, band_2, band_3 = in_image.split()
    if genetic_parameters.edge_method == "I":
        edge_raw = band_1
    else:
        edge_raw = band_2
    return edge_raw

def low_pass(source_image, image_number):
    lp_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    si_array = array.array("f")
    si_array.fromlist(list(source_image.getdata()))
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            if lcv_row == 0 or lcv_row == images.y_dim[image_number] - 1 or lcv_column == 0 or lcv_column ==
images.x_dim[image_number] - 1:
                value = si_array[lcv_row * x_dim + lcv_column]
                lp_image.putpixel((lcv_column, lcv_row), value)
            else:
                value = si_array[lcv_row * x_dim + lcv_column]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column - 1)]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column)]
                value = value + si_array[((lcv_row - 1) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column - 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column - 1))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column + 1))]
                value = float(value)/float(9)
                lp_image.putpixel((lcv_column, lcv_row), value)

```

```

return lp_image

def calculate_dog(SD):
    multipliers = []
    pi = math.pi
    sqrt2 = math.sqrt(2)
    for lcv_x in range(-5, 6):
        lcv_calculate = float(lcv_x)
        term_1 = -0.5 * sqrt2 / (math.sqrt(pi) * math.pow(SD, 3))
        term_2 = math.exp(-0.5 * ((lcv_calculate * lcv_calculate) / (SD * SD)))
        term_3 = 0.5 * math.sqrt(2) / (math.sqrt(pi) * math.pow(SD, 5))
        term_4 = (lcv_calculate) * (lcv_calculate)
        term_5 = math.exp(-0.5 * (lcv_calculate * lcv_calculate) / (SD * SD))
        coef = (term_1 * term_2) + (term_3 * term_4 * term_5)
        multipliers.append(coef)

    return multipliers

def lmin(edge_raw, image_number, SD):
    lm_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    lmax_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    multipliers = calculate_dog(SD)
    edge_array = array.array("f")
    edge_array.fromlist(list(edge_raw.getdata()))

    pp_lm = lm_image.putpixel
    pp_lmax = lmax_image.putpixel

    for lcv_row in range (6, images.y_dim[image_number] - 6):
        for lcv_column in range (6, images.x_dim[image_number] - 6):
            value_01 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[lcv_column-1 + (lcv_row + 1) * x_dim] +

```

```

edge_array[lcv_column+1 + (lcv_row - 1) * x_dim]
    value_1 = edge_array[(lcv_column-5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column-6) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row - 6) * x_dim]
    value_2 = edge_array[(lcv_column-4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column-5) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row - 5) * x_dim]
    value_3 = edge_array[(lcv_column-3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column-4) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row - 4) * x_dim]
    value_4 = edge_array[(lcv_column-2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column-3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row - 3) * x_dim]
    value_5 = edge_array[(lcv_column-1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column-2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row - 2) * x_dim]
    value_6 = edge_array[(lcv_column+1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
    value_7 = edge_array[(lcv_column+2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
    value_8 = edge_array[(lcv_column+3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 2) * x_dim]
    value_9 = edge_array[(lcv_column+4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 3) * x_dim]
    value_10 = edge_array[(lcv_column+5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row + 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row + 4) * x_dim]
    value_11 = edge_array[(lcv_column) + (lcv_row-5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 5) * x_dim]
    value_12 = edge_array[(lcv_column) + (lcv_row-4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 4) * x_dim]
    value_13 = edge_array[(lcv_column) + (lcv_row-3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 3) * x_dim]
    value_14 = edge_array[(lcv_column) + (lcv_row-2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 2) * x_dim]
    value_15 = edge_array[(lcv_column) + (lcv_row-1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 1) * x_dim]
    value_02 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row) * x_dim]
    value_16 = edge_array[(lcv_column) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
    value_17 = edge_array[(lcv_column) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 2) * x_dim] +

```

```

edge_array[lcv_column + 1 + (lcv_row + 2) * x_dim]
    value_18 = edge_array[(lcv_column) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 3) * x_dim]
    value_19 = edge_array[(lcv_column) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 4) * x_dim]
    value_20 = edge_array[(lcv_column) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 5) * x_dim]
    value_21 = edge_array[(lcv_column-5) + (lcv_row) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 5 + (lcv_row + 1) * x_dim]
    value_22 = edge_array[(lcv_column-4) + (lcv_row) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 1) * x_dim]
    value_23 = edge_array[(lcv_column-3) + (lcv_row) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 1) * x_dim]
    value_24 = edge_array[(lcv_column-2) + (lcv_row) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 1) * x_dim]
    value_25 = edge_array[(lcv_column-1) + (lcv_row) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 1) * x_dim]
    value_03 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + (lcv_row + 1) * x_dim]
    value_26 = edge_array[(lcv_column+1) + (lcv_row) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
    value_27 = edge_array[(lcv_column+2) + (lcv_row) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row + 1) * x_dim]
    value_28 = edge_array[(lcv_column+3) + (lcv_row) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
    value_29 = edge_array[(lcv_column+4) + (lcv_row) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 1) * x_dim]
    value_30 = edge_array[(lcv_column+5) + (lcv_row) * x_dim] + edge_array[(lcv_column + 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 1) * x_dim]
    value_31 = edge_array[(lcv_column-5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 6) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 6) * x_dim]
    value_32 = edge_array[(lcv_column-4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 5) * x_dim]
    value_33 = edge_array[(lcv_column-3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 4) * x_dim]
    value_34 = edge_array[(lcv_column-2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row + 1) * x_dim] +

```

```

edge_array[lcv_column - 1 + (lcv_row + 3) * x_dim]
    value_35 = edge_array[(lcv_column-1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row + 2) * x_dim]
    value_04 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
    value_36 = edge_array[(lcv_column+1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
    value_37 = edge_array[(lcv_column+2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row - 1) * x_dim]
    value_38 = edge_array[(lcv_column+3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row - 2) * x_dim]
    value_39 = edge_array[(lcv_column+4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row - 3) * x_dim]
    value_40 = edge_array[(lcv_column+5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row - 4) * x_dim]

```

```

lm_list = []

```

```

    lm_1 = multipliers[0]*value_1 + multipliers[1]*value_2 + multipliers[2]*value_3 + multipliers[3]*value_4 + multipliers[4] *
value_5 + multipliers[5] * value_01 + multipliers[6] * value_6 + multipliers[7]*value_7 + multipliers[8]*value_8 + multipliers[9] * value_9 + multipliers[10]
* value_10

```

```

    lm_2 = multipliers[0]*value_11 + multipliers[1]*value_12 + multipliers[2]*value_13 + multipliers[3]*value_14 + multipliers[4] *
value_15 + multipliers[5] * value_02 + multipliers[6] * value_16 + multipliers[7]*value_17 + multipliers[8]*value_18 + multipliers[9] * value_19 +
multipliers[10] * value_20

```

```

    lm_3 = multipliers[0]*value_21 + multipliers[1]*value_22 + multipliers[2]*value_23 + multipliers[3]*value_24 + multipliers[4] *
value_25 + multipliers[5] * value_03 + multipliers[6] * value_26 + multipliers[7]*value_27 + multipliers[8]*value_28 + multipliers[9] * value_29 +
multipliers[10] * value_30

```

```

    lm_4 = multipliers[0]*value_31 + multipliers[1]*value_32 + multipliers[2]*value_33 + multipliers[3]*value_34 + multipliers[4] *
value_35 + multipliers[5] * value_04 + multipliers[6] * value_36 + multipliers[7]*value_37 + multipliers[8]*value_38 + multipliers[9] * value_39 +
multipliers[10] * value_40

```

```

    lm_list.append(lm_1)

```

```

    lm_list.append(lm_2)

```

```

    lm_list.append(lm_3)

```

```

    lm_list.append(lm_4)

```

```

lm_list.sort()

```

```

pp_lm((lcv_column, lcv_row), lm_list[3])

```

```

        pp_lmax((lcv_column, lcv_row), lm_list[0])
    return lm_image, lmax_image

def make_int_maps(image_number):
    images.value_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    images.radius_image = Image.new("L", (images.x_dim[image_number], images.y_dim[image_number]), 0)

def find_value_diam(image_number, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay):

    if min_total < 0:
        min_total = 0

    edge_array = array.array("f")
    edge_array.fromlist(list(images.edge_images[image_number].getdata()))

    lmin_array = array.array("f")
    lmin_array.fromlist(list(images.lmin_image.getdata()))

    ratio_array = array.array("f")
    ratio_array.fromlist(list(images.ratio_images[image_number].getdata()))

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

    pp_value = images.value_image.putpixel
    pp_radius = images.radius_image.putpixel

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            test_1 = lmin_array[lcv_line * x_dim + lcv_column]
            test_2 = ratio_array[lcv_line * x_dim + lcv_column]
            if test_1 < center_thresh and test_2 > ratio_thresh:
                high_value = -999
                high_radius = 0
                stop = "N"
                stop_now = "N"

```

```

for lcv_radius in range(genetic_parameters.min_radius, genetic_parameters.max_radius + 1):
    if stop == "Y":
        stop_now = "Y"
    pixels = 0
    value = 0
    sqrad = lcv_radius * lcv_radius
    for lcv_y in range(-1 * lcv_radius, lcv_radius + 1):
        x_radius = int(round(math.sqrt(sqrad - lcv_y * lcv_y)))
        position_y = lcv_line + lcv_y
        pos_position_x = lcv_column + x_radius
        neg_position_x = lcv_column - x_radius
        if ratio_array[position_y * x_dim + pos_position_x] < adj_thresh or edge_array[position_y * x_dim +
pos_position_x] < ae_thresh:
            stop = "Y"
        if ratio_array[position_y * x_dim + neg_position_x] < adj_thresh or edge_array[position_y * x_dim +
neg_position_x] < ae_thresh:
            stop = "Y"
        value = value + lmin_array[position_y * x_dim + pos_position_x]
        value = value + lmin_array[position_y * x_dim + neg_position_x]
        if x_radius == 0:
            pixels = pixels + 1
        else:
            pixels = pixels + 2

    total_value = float(value) / float(pixels)
    total_value = total_value + decay*lcv_radius
    if total_value > high_value:
        high_value = total_value
        high_radius = lcv_radius
    if high_value < min_total:
        high_value = 0
    if stop_now == "Y":
        break

pp_value((lcv_column, lcv_line), high_value)
pp_radius((lcv_column, lcv_line), high_radius)

```



```

def refine_trees(image_number, alpha, beta, gamma):

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

    lmax_array = array.array("f")
    lmax_array.fromlist(list(images.lmax_image.getdata()))

    radius_array = array.array("f")
    radius_array.fromlist(list(images.radius_image.getdata()))

    value_array = array.array("f")
    value_array.fromlist(list(images.value_image.getdata()))

    ratio_array = array.array("f")
    ratio_array.fromlist(list(images.ratio_images[image_number].getdata()))

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
            if pixel_rad > 0:
                keep = 1
                score = alpha * value_array[lcv_line * x_dim + lcv_column]
                score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
                score = score + gamma * ratio_array[lcv_line * x_dim + lcv_column]
                for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                    for win_x in range (lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                        radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                        ignore = "N"
                        if radius > pixel_rad:
                            ignore = "Y"
                        if ignore == "N":
                            wscore = alpha * value_array[win_y * x_dim + win_x]
                            wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]

```

```

        wscore = wscore + gamma * ratio_array[win_y * x_dim + win_x]
        if wscore > score:
            keep = 0
            break
    if int(keep) == 0:
        break
images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

radius_array = array.array("f")
radius_array.fromlist(list(images.radius_image.getdata()))

#The following routine cleans up any small trees that are encroaching on a large tree's space

for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_array[lcv_line * x_dim + lcv_column]
            score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
            score = score + gamma * ratio_array[lcv_line * x_dim + lcv_column]
            for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                    radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                    ignore = "N"
                    if radius > pixel_rad:
                        ignore = "Y"
                    if ignore == "N" and radius_array[win_y * x_dim + win_x] > 0:
                        wscore = alpha * value_array[win_y * x_dim + win_x]
                        wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
                        wscore = wscore + gamma * ratio_array[win_y * x_dim + win_x]
                        if wscore < score:
                            images.radius_image.putpixel((win_x, win_y), 0)

```

```

def make_output(image_number):
    trees = 0
    for lcv_line in range (genetic_parameters.buffer, images.y_dim[image_number] - genetic_parameters.buffer):
        for lcv_column in range (genetic_parameters.buffer, images.x_dim[image_number] - genetic_parameters.buffer):
            pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))
            center_x = float(images.x_dim[image_number]) / float(2)
            center_y = float(images.y_dim[image_number]) / float(2)
            term_1 = (center_x - float(lcv_column)) * (center_x - float(lcv_column))
            term_2 = (center_y - float(lcv_line)) * (center_y - float(lcv_line))
            distance = math.sqrt(term_1 + term_2)
            if pixel_rad > 0 and int(round(distance)) <= genetic_parameters.plot_radius:
                trees = trees + 1
    return trees

def start(image_number, first_run, alpha, beta, gamma, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay, SD):
    if first_run[image_number] == "Y":
        ratio(image_number)
    if first_run[image_number] == "Y":
        images.lp_images.append(low_pass(images.ratio_images[image_number], image_number))
    if first_run[image_number] == "Y":
        if genetic_parameters.edge_method == "R":
            images.lp_images.append(lp_image)
        else:
            edge_raw = edge(image_number)
            images.edge_images.append(low_pass(edge_raw, image_number))
    else:
        edge_raw = edge(image_number)
    images.lmin_image, images.lmax_image = lmin(edge_raw, image_number, SD)
    make_int_maps(image_number)
    find_value_diam(image_number, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay)
    refine_trees(image_number, alpha, beta, gamma)
    trees = make_output(image_number)
    del images.radius_image
    del images.value_image

```

```

        return trees

def begin():

#
#This function will start the program out if it is being run as a stand alone program
#

    ofile = open("dummy.txt", "w")

    output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"

    ofile.write(output_string)

    ofile.write(output_string)

    first_run = []
    images.ratio_images = []
    images.lmin_images = []
    images.lmax_images = []
    images.x_dim = []
    images.y_dim = []
    images.lp_images = []
    images.edge_images = []

    genetic_parameters.plot_radius = 30

genetic_parameters.iterations = 5000

genetic_parameters.population_size = 100

    genetic_parameters.mutation_strength = 0.3

genetic_parameters.number_test_images = 25

    for lcv_first_run in range(0, genetic_parameters.number_test_images):

```

```
first_run.append("Y")

genetic_parameters.output_file = "c:/temp/tree_tops/pca_3_results.txt"

genetic_parameters.image_names = []

genetic_parameters.image_names.append("1_test.tif")
genetic_parameters.image_names.append("2_test.tif")
genetic_parameters.image_names.append("3_test.tif")
genetic_parameters.image_names.append("4_test.tif")
genetic_parameters.image_names.append("5_test.tif")
genetic_parameters.image_names.append("6_test.tif")
genetic_parameters.image_names.append("7_test.tif")
genetic_parameters.image_names.append("8_test.tif")
genetic_parameters.image_names.append("9_test.tif")
genetic_parameters.image_names.append("10_test.tif")
genetic_parameters.image_names.append("11_test.tif")
genetic_parameters.image_names.append("12_test.tif")
genetic_parameters.image_names.append("13_test.tif")
genetic_parameters.image_names.append("14_test.tif")
genetic_parameters.image_names.append("15_test.tif")
genetic_parameters.image_names.append("16_test.tif")
genetic_parameters.image_names.append("17_test.tif")
genetic_parameters.image_names.append("18_test.tif")
genetic_parameters.image_names.append("19_test.tif")
genetic_parameters.image_names.append("20_test.tif")
genetic_parameters.image_names.append("21_test.tif")
genetic_parameters.image_names.append("22_test.tif")
genetic_parameters.image_names.append("23_test.tif")
genetic_parameters.image_names.append("24_test.tif")
genetic_parameters.image_names.append("25_test.tif")

genetic_parameters.buffer = 10
```

```

genetic_parameters.ratio_method = "C"

genetic_parameters.edge_method = "T"

genetic_parameters.filter = "L"

genetic_parameters.min_radius = 1

genetic_parameters.max_radius = 8

genetic_parameters.number_of_parameters = 10

alpha = 4.28201496704
beta = -2.8870125893
gamma = 0.290424412053
center_thresh = 10.558698599
ratio_thresh = 3.12021303335
adj_thresh = -4.58946182298
ae_thresh = 67.4823676542
min_total = 2.98744536708
decay = -17.8573542611
SD = 2.34220220579

for lcv_image in range(0, genetic_parameters.number_test_images):
    trees = start(lcv_image, first_run, alpha, beta, gamma, center_thresh, ratio_thresh, adj_thresh, ae_thresh, min_total, decay, SD)

    print "Stand:", lcv_image, "Tree count =", trees

if __name__ == "__main__":
    begin()

```

Appendix D. The program for setting parameters for the program that locates trees in LiDAR images and simultaneously determines biomass.

D.1 Main program

```
from Tkinter import *
import math
import random
import Image
import sys
import string
import genetic_parameters
import images
import gc
import array
import nelder_biomass_score

def raw(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    x_dim, y_dim = in_image.size
    images.x_dim.append(x_dim)
    images.y_dim.append(y_dim)
    images.raw_images.append(in_image)

def edge(image_number):
    edge_raw = Image.open(genetic_parameters.image_names[image_number])
    return edge_raw

def low_pass(source_image, image_number):
    lp_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    si_array = array.array("d")
    si_array.fromlist(list(source_image.getdata()))
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    for lcv_row in range(0, y_dim):
```

```

        for lcv_column in range(0, x_dim):
            if lcv_row == 0 or lcv_row == images.y_dim[image_number] - 1 or lcv_column == 0 or lcv_column ==
images.x_dim[image_number] - 1:
                value = si_array[lcv_row * x_dim + lcv_column]
                lp_image.putpixel((lcv_column, lcv_row), value)
            else:
                value = si_array[lcv_row * x_dim + lcv_column]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column - 1)]
                value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column)]
                value = value + si_array[((lcv_row - 1) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column - 1))]
                value = value + si_array[((lcv_row) * x_dim + (lcv_column + 1))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column - 1))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column))]
                value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column + 1))]
                value = float(value)/float(9)
                lp_image.putpixel((lcv_column, lcv_row), value)

    return lp_image

def calculate_dog(SD):
    multipliers = []
    pi = math.pi
    sqrt2 = math.sqrt(2)
    for lcv_x in range(-5, 6):
        lcv_calculate = float(lcv_x)
        term_1 = -0.5 * sqrt2 / (math.sqrt(pi) * math.pow(SD, 3))
        term_2 = math.exp(-0.5 * ((lcv_calculate * lcv_calculate) / (SD * SD)))
        term_3 = 0.5 * math.sqrt(2) / (math.sqrt(pi) * math.pow(SD, 5))
        term_4 = (lcv_calculate) * (lcv_calculate)
        term_5 = math.exp(-0.5 * (lcv_calculate * lcv_calculate) / (SD * SD))
        coef = (term_1 * term_2) + (term_3 * term_4 * term_5)
        multipliers.append(coef)

    return multipliers

```



```

def lmin(edge_raw, image_number, SD):
    lm_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    lmax_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    multipliers = calculate_dog(SD)
    #multipliers = [0, 1, 1, 1, 1, -8, 1, 1, 1, 1, 0]
    edge_array = array.array("f")
    edge_array.fromlist(list(edge_raw.getdata()))

    pp_lm = lm_image.putpixel
    pp_lmax = lmax_image.putpixel

    for lcv_row in range (6, images.y_dim[image_number] - 6):
        for lcv_column in range (6, images.x_dim[image_number] - 6):
            value_01 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[lcv_column-1 + (lcv_row + 1) * x_dim] +
edge_array[lcv_column+1 + (lcv_row - 1) * x_dim]
            value_1 = edge_array[(lcv_column-5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column-6) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row - 6) * x_dim]
            value_2 = edge_array[(lcv_column-4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column-5) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row - 5) * x_dim]
            value_3 = edge_array[(lcv_column-3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column-4) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row - 4) * x_dim]
            value_4 = edge_array[(lcv_column-2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column-3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row - 3) * x_dim]
            value_5 = edge_array[(lcv_column-1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column-2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row - 2) * x_dim]
            value_6 = edge_array[(lcv_column+1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
            value_7 = edge_array[(lcv_column+2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
            value_8 = edge_array[(lcv_column+3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 2) * x_dim]
            value_9 = edge_array[(lcv_column+4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 3) * x_dim]

```

```

        value_10 = edge_array[(lcv_column+5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row + 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row + 4) * x_dim]
        value_11 = edge_array[(lcv_column) + (lcv_row-5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 5) * x_dim]
        value_12 = edge_array[(lcv_column) + (lcv_row-4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 4) * x_dim]
        value_13 = edge_array[(lcv_column) + (lcv_row-3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 3) * x_dim]
        value_14 = edge_array[(lcv_column) + (lcv_row-2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 2) * x_dim]
        value_15 = edge_array[(lcv_column) + (lcv_row-1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 1) * x_dim]
        value_02 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row) * x_dim]
        value_16 = edge_array[(lcv_column) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
        value_17 = edge_array[(lcv_column) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 2) * x_dim]
        value_18 = edge_array[(lcv_column) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 3) * x_dim]
        value_19 = edge_array[(lcv_column) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 4) * x_dim]
        value_20 = edge_array[(lcv_column) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 5) * x_dim]
        value_21 = edge_array[(lcv_column-5) + (lcv_row) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 5 + (lcv_row + 1) * x_dim]
        value_22 = edge_array[(lcv_column-4) + (lcv_row) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 1) * x_dim]
        value_23 = edge_array[(lcv_column-3) + (lcv_row) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 1) * x_dim]
        value_24 = edge_array[(lcv_column-2) + (lcv_row) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 1) * x_dim]
        value_25 = edge_array[(lcv_column-1) + (lcv_row) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 1) * x_dim]
        value_03 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + (lcv_row + 1) * x_dim]

```

```

value_26 = edge_array[(lcv_column+1) + (lcv_row) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
value_27 = edge_array[(lcv_column+2) + (lcv_row) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row + 1) * x_dim]
value_28 = edge_array[(lcv_column+3) + (lcv_row) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
value_29 = edge_array[(lcv_column+4) + (lcv_row) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 1) * x_dim]
value_30 = edge_array[(lcv_column+5) + (lcv_row) * x_dim] + edge_array[(lcv_column + 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 1) * x_dim]
value_31 = edge_array[(lcv_column-5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 6) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 6) * x_dim]
value_32 = edge_array[(lcv_column-4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 5) * x_dim]
value_33 = edge_array[(lcv_column-3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 4) * x_dim]
value_34 = edge_array[(lcv_column-2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 3) * x_dim]
value_35 = edge_array[(lcv_column-1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row + 2) * x_dim]
value_04 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
value_36 = edge_array[(lcv_column+1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
value_37 = edge_array[(lcv_column+2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row - 1) * x_dim]
value_38 = edge_array[(lcv_column+3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row - 2) * x_dim]
value_39 = edge_array[(lcv_column+4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row - 3) * x_dim]
value_40 = edge_array[(lcv_column+5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row - 4) * x_dim]

```

```
lm_list = []
```

```
lm_1 = multipliers[0]*value_1 + multipliers[1]*value_2 + multipliers[2]*value_3 + multipliers[3]*value_4 + multipliers[4] *
```

```

value_5 + multipliers[5] * value_01 + multipliers[6] * value_6 + multipliers[7]*value_7 + multipliers[8]*value_8 + multipliers[9] * value_9 + multipliers[10]
* value_10
        lm_2 = multipliers[0]*value_11 + multipliers[1]*value_12 + multipliers[2]*value_13 + multipliers[3]*value_14 + multipliers[4] *
value_15 + multipliers[5] * value_02 + multipliers[6] * value_16 + multipliers[7]*value_17 + multipliers[8]*value_18 + multipliers[9] * value_19 +
multipliers[10] * value_20
        lm_3 = multipliers[0]*value_21 + multipliers[1]*value_22 + multipliers[2]*value_23 + multipliers[3]*value_24 + multipliers[4] *
value_25 + multipliers[5] * value_03 + multipliers[6] * value_26 + multipliers[7]*value_27 + multipliers[8]*value_28 + multipliers[9] * value_29 +
multipliers[10] * value_30
        lm_4 = multipliers[0]*value_31 + multipliers[1]*value_32 + multipliers[2]*value_33 + multipliers[3]*value_34 + multipliers[4] *
value_35 + multipliers[5] * value_04 + multipliers[6] * value_36 + multipliers[7]*value_37 + multipliers[8]*value_38 + multipliers[9] * value_39 +
multipliers[10] * value_40
        lm_list.append(lm_1)
        lm_list.append(lm_2)
        lm_list.append(lm_3)
        lm_list.append(lm_4)
    lm_list.sort()
    pp_lm((lcv_column, lcv_row), lm_list[3])
        pp_lmax((lcv_column, lcv_row), lm_list[0])
return lm_image, lmax_image

def make_int_maps(image_number):
    images.value_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    images.radius_image = Image.new("L", (images.x_dim[image_number], images.y_dim[image_number]), 0)

def find_value_diam(image_number, center_thresh, ratio_thresh, ae_thresh, min_total, decay):

    if min_total < 0:
        min_total = 0

    edge_array = array.array("f")
    edge_array.fromlist(list(images.edge_images[image_number].getdata()))

    lmin_array = array.array("f")
    lmin_array.fromlist(list(images.lmin_image.getdata()))

    raw_array = array.array("f")

```

```

raw_array.fromlist(list(images.raw_images[image_number].getdata()))

x_dim = images.x_dim[image_number]
y_dim = images.y_dim[image_number]

pp_value = images.value_image.putpixel
pp_radius = images.radius_image.putpixel

for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        test_1 = lmin_array[lcv_line * x_dim + lcv_column]
        test_2 = raw_array[lcv_line * x_dim + lcv_column]
        if test_1 < center_thresh and test_2 > ratio_thresh:
            high_value = -999
            high_radius = 0
            stop = "N"
            stop_now = "N"
            for lcv_radius in range(genetic_parameters.min_radius, genetic_parameters.max_radius + 1):
                if stop == "Y":
                    stop_now = "Y"
                pixels = 0
                value = 0
                sqrad = lcv_radius * lcv_radius
                for lcv_y in range (-1 * lcv_radius, lcv_radius + 1):
                    x_radius = int(round(math.sqrt(sqrad - lcv_y * lcv_y)))
                    position_y = lcv_line + lcv_y
                    pos_position_x = lcv_column + x_radius
                    neg_position_x = lcv_column - x_radius
                    if (test_2 - edge_array[position_y * x_dim + pos_position_x]) > ae_thresh:
                        stop = "Y"
                    if (test_2 - edge_array[position_y * x_dim + neg_position_x]) > ae_thresh:
                        stop = "Y"
                    value = value + lmin_array[position_y * x_dim + pos_position_x]
                    value = value + lmin_array[position_y * x_dim + neg_position_x]
                    if x_radius == 0:
                        pixels = pixels + 1

```

```

        else:
            pixels = pixels + 2

    total_value = float(value) / float(pixels)
    total_value = total_value + decay*lcv_radius
    if total_value > high_value:
        high_value = total_value
        high_radius = lcv_radius
    if high_value < min_total:
        high_value = 0
    if stop_now == "Y":
        break

    pp_value((lcv_column, lcv_line), high_value)
    pp_radius((lcv_column, lcv_line), high_radius)

```

```
def refine_trees(image_number, alpha, beta, gamma):
```

```

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

```

```

    lmax_array = array.array("f")
    lmax_array.fromlist(list(images.lmax_image.getdata()))

```

```

    radius_array = array.array("f")
    radius_array.fromlist(list(images.radius_image.getdata()))

```

```

    value_array = array.array("f")
    value_array.fromlist(list(images.value_image.getdata()))

```

```

    raw_array = array.array("f")
    raw_array.fromlist(list(images.raw_images[image_number].getdata()))

```

```

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):

```

```

pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
if pixel_rad > 0:
    keep = 1
    score = alpha * value_array[lcv_line * x_dim + lcv_column]
    score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
    score = score + gamma * raw_array[lcv_line * x_dim + lcv_column]
    for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
        for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
            radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
            ignore = "N"
            if round(radius) > pixel_rad:
                ignore = "Y"
            if ignore == "N":
                wscore = alpha * value_array[win_y * x_dim + win_x]
                wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
                wscore = wscore + gamma * raw_array[win_y * x_dim + win_x]
                if wscore > score:
                    keep = 0
                    break
        if int(keep) == 0:
            break
    images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

```

```

radius_array = array.array("f")
radius_array.fromlist(list(images.radius_image.getdata()))

```

```

for lcv_line in range(genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range(genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_array[lcv_line * x_dim + lcv_column]
            score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
            score = score + gamma * raw_array[lcv_line * x_dim + lcv_column]
            for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):

```

```

radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
ignore = "N"
if round(radius) > pixel_rad:
    ignore = "Y"
if ignore == "N" and radius_array[win_y * x_dim + win_x] > 0:
    wscore = alpha * value_array[win_y * x_dim + win_x]
    wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
    wscore = wscore + gamma * raw_array[win_y * x_dim + win_x]
    if wscore < score:
        images.radius_image.putpixel((win_x, win_y), 0)

```

```

def find_height(lcv_column, lcv_line, pixel_rad, image_number):
    max_height = -2000
    for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
        for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
            radius = math.sqrt((win_x - lcv_column) * (win_x - lcv_column) + (win_y - lcv_line) * (win_y - lcv_line))
            if round(radius) <= pixel_rad:
                lidar_value = images.raw_images[image_number].getpixel((win_x, win_y))
                if lidar_value > max_height:
                    max_height = lidar_value

    if max_height == -2000:
        print pixel_rad
    return max_height

```

```

def make_output(image_number):
    trees = 0
    diameter_list = []
    height_list = []
    for lcv_line in range(genetic_parameters.buffer, images.y_dim[image_number] - genetic_parameters.buffer):
        for lcv_column in range(genetic_parameters.buffer, images.x_dim[image_number] - genetic_parameters.buffer):
            pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))
            center_x = float(images.x_dim[image_number]) / float(2)
            center_y = float(images.y_dim[image_number]) / float(2)

```



```

term_1 = (center_x - float(lcv_column)) * (center_x - float(lcv_column))
term_2 = (center_y - float(lcv_line)) * (center_y - float(lcv_line))
distance = math.sqrt(term_1 + term_2)
if pixel_rad > 0 and int(round(distance)) <= genetic_parameters.plot_radius:
    trees = trees + 1
    diameter = genetic_parameters.pixel_length + 2 * genetic_parameters.pixel_length * pixel_rad
    diameter_list.append(diameter)
    height = find_height(lcv_column, lcv_line, pixel_rad, image_number)
    height_list.append(height)
area = (math.pi * math.pow(genetic_parameters.plot_radius * genetic_parameters.pixel_length, 2)) / 10000
density = trees / area
genetic_parameters.tree_count.append(trees)
genetic_parameters.tree_density.append(density)
genetic_parameters.crown_widths.append(diameter_list)
genetic_parameters.tree_heights.append(height_list)

def start(image_number, alpha, beta, gamma, center_thresh, ratio_thresh, ae_thresh, min_total, decay, SD):

    genetic_parameters.counter = genetic_parameters.counter + 1
    if genetic_parameters.first_run[image_number] == "Y":
        raw(image_number)
    if genetic_parameters.first_run[image_number] == "Y":
        images.lp_images.append(low_pass(images.raw_images[image_number], image_number))
    if genetic_parameters.first_run[image_number] == "Y":
        if genetic_parameters.edge_method == "R":
            images.edge_images.append(lp_image)
        else:
            edge_raw = edge(image_number)
            images.edge_images.append(low_pass(edge_raw, image_number))
    else:
        edge_raw = edge(image_number)
    images.lmin_image, images.lmax_image = lmin(edge_raw, image_number, SD)
    make_int_maps(image_number)
    find_value_diam(image_number, center_thresh, ratio_thresh, ae_thresh, min_total, decay)
    refine_trees(image_number, alpha, beta, gamma)
    make_output(image_number)

```

```

del images.radius_image
del images.value_image

#
# Nelder and Meade Simplex Algorithm
#

def eval_func (parameter_list):

    genetic_parameters.crown_widths = []
    genetic_parameters.tree_density = []
    genetic_parameters.tree_count = []
    genetic_parameters.tree_heights = []

    for lcv_image in range (0, genetic_parameters.number_test_images):
        alpha = parameter_list[0]
        beta = parameter_list[1]
        gamma = parameter_list[2]
        center = parameter_list[3]
        ratio = parameter_list[4]
        ae = parameter_list[5]
        min = parameter_list[6]
        decay = parameter_list[7]
        SD = parameter_list[8]
        start(lcv_image, alpha, beta, gamma, center, ratio, ae, min, decay, SD)
    genetic_parameters.first_run[lcv_image] = "N"

    biomass_score = nelder_biomass_score.nelder_main([-0.69113, 3.31007, 0.85506])

    genetic_parameters.print_biomass_estimates = "N"

    tree_score = 0

```

```

for lcv_image in range(0, genetic_parameters.number_test_images):
    adjusted_tree_count = (genetic_parameters.tree_count[lcv_image] + genetic_parameters.offset) * genetic_parameters.gain
    adjusted_actual_trees = (genetic_parameters.actual_tree_count[lcv_image] + genetic_parameters.offset) * genetic_parameters.gain
    tree_count_error = adjusted_tree_count - adjusted_actual_trees
    tree_score = tree_score + tree_count_error * tree_count_error

score = (tree_score * genetic_parameters.relative_importance + biomass_score) / float(1000)

if genetic_parameters.verbose == "Y":
    print "Verbose values"
    print parameter_list
    print tree_score
    print biomass_score

return score

def create_initial_values(N, a, initial_parameter_list):

    sqrt2 = math.pow(2, 0.5)
    sqrtNp1 = math.pow(N+1, 0.5)

    parameter_matrix = []

    parameter_matrix.append(initial_parameter_list)

    for lcv_y in range(1, N+1):
        parameter_row = array.array("f", [])
        for lcv_x in range(0, N):
            p = (a[lcv_x] / (N * sqrt2)) * (sqrtNp1 + N - 1)
            q = (a[lcv_x] / (N * sqrt2)) * (sqrtNp1 - 1)
            if lcv_y == lcv_x + 1:
                value_to_add = initial_parameter_list[lcv_x] + p
            else:
                value_to_add = initial_parameter_list[lcv_x] + q
            parameter_row.append(value_to_add)

```

```

        parameter_matrix.append(parameter_row)

score_list = []

for lcv_y in range (0, N+1):
    score = eval_func(parameter_matrix[lcv_y])
    score_list.append(score)

print "Score for point 0:", score_list[0]
print "Parameters for point 0:", parameter_matrix[0]

return parameter_matrix, score_list

def calculate_reflected(N, parameter_matrix, score_list, alpha):

    #
    # Find the parameter value and number of the worst point
    #

    worst_score = score_list[0]
    worst_point = 0

    #
    # Find the worst score and the number of the point with the worst score
    #

    for lcv in range (1, N+1):
        if score_list[lcv] > worst_score:
            worst_point = lcv
            worst_score = score_list[lcv]

    #
    # Find the centroids of all but the worst point
    #

```

```

centroid_list = []

for lcv_x in range(0, N):
    parameter_sum = 0
    for lcv_y in range(0, N+1):
        parameter_sum = parameter_sum + parameter_matrix[lcv_y][lcv_x]
    centroid = (float(1) / float(N))* (parameter_sum - parameter_matrix[worst_point][lcv_x])
    centroid_list.append(centroid)

#
# Find the parameters for the reflected point
#

reflected_parameters = []
worst_list = []

for lcv in range (0, N):
    reflected_value = centroid_list[lcv] + alpha * (centroid_list[lcv] - parameter_matrix[worst_point][lcv])
    reflected_parameters.append(reflected_value)
    worst_list.append(parameter_matrix[worst_point][lcv])

return reflected_parameters, centroid_list, worst_list, worst_point, worst_score

def calculate_contracted(N, reflected_parameters, worst_parameters, centroid_list, was_it_the_worst, beta):

#
# Calculates the value of a contracted point
#

contracted_parameters = []

if was_it_the_worst == "yes":
    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - worst_parameters[lcv])
        contracted_parameters.append(new_parameter_value)

else:

```

```

    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - reflected_parameters[lcv])
        contracted_parameters.append(new_parameter_value)

return contracted_parameters

```

```

def calculate_expansion(N, reflected_parameters, centroid_list, gamma):

```

```

    #
    # Calculates the value of an expansion point
    #

    expansion_parameters = []

    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] + gamma * (reflected_parameters[lcv] - centroid_list[lcv])
        expansion_parameters.append(new_parameter_value)

    return expansion_parameters

```

```

def move_incrementally(N, parameter_matrix, score_list):

```

```

    #
    # Moves the points in the parameter matrix incrementally closer to the parameters of the best member of the population
    #

    best_score = score_list[0]
    best_score_number = 0

    for lcv in range (1, N+1):
        if score_list[lcv] < best_score:
            best_score = score_list[lcv]
            best_score_number = lcv

```

```

for lcv_x in range(0, N):
    for lcv_y in range(0, N+1):
        moved_point = (parameter_matrix[lcv_y][lcv_x] + parameter_matrix[best_score_number][lcv_x]) / float(2)
        parameter_matrix[lcv_y][lcv_x] = moved_point
score_list = array.array('f', [])
for lcv in range(0, N+1):
    score = eval_func(parameter_matrix[lcv])
    score_list.append(score)

return parameter_matrix, score_list

def find_best_score(N, score_list):
    print "Score 0 = ", score_list[0]
    best_score = score_list[0]
    best_score_index = 0
    for lcv in range (1, N+1):
        if score_list[lcv] < best_score:
            best_score = score_list[lcv]
            best_score_index = lcv

    return best_score, best_score_index

def check_for_stop(N, score_list):
    first_score = int(10 * score_list[0])
    stop = "yes"
    for lcv in range(1, N+1):
        if first_score <> int(10 * score_list[lcv]):
            stop = "no"
    return stop

def replace_point(N, worst_point, best_parameters, best_score, parameter_matrix, score_list):
    parameter_matrix[worst_point] = best_parameters
    score_list[worst_point] = best_score
    return parameter_matrix, score_list

```

```

def nelder_main():

    #
    # The main engine for the algorithm
    #

    genetic_parameters.verbose = "N"

    print "Up here..."

    ofile = open(genetic_parameters.output_file, "w")

    output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"

    ofile.write(output_string)

    ofile.write(output_string)

    images.raw_images = []
    images.lmin_images = []
    images.lmax_images = []
    images.x_dim = []
    images.y_dim = []
    images.lp_images = []
    images.edge_images = []

    genetic_parameters.best_equation_parameter_list = []

    genetic_parameters.first_run = []

    for lcv in range (0, genetic_parameters.number_test_images):
        genetic_parameters.first_run.append("Y")

    N = genetic_parameters.number_of_parameters
    a = [5,5,5,10,10,5,10,30,1.5]
    alpha = 1.0

```



```

beta = 0.5
gamma = 2.0
acc = 0.0001
initial_parameter_list = genetic_parameters.initial_parameter_list
score = eval_func(initial_parameter_list)

max_iter = 1000
stop = "no"
current_iter = 0

parameter_matrix, score_list = create_initial_values(N, a, initial_parameter_list)

score_array = []

while (current_iter <= max_iter) and (stop == "no"):
    reflected_parameters, centroid_list, worst_list, worst_point, worst_score = calculate_reflected(N, parameter_matrix, score_list, alpha)
    reflected_score = eval_func(reflected_parameters)
    best_score, best_score_index = find_best_score(N, score_list)
    score_array.append(best_score)
    print "Best score = ", best_score
    print "Best score index = ", best_score_index
    print "Current iteration = ", current_iter
    print "-----"

    if reflected_score > worst_score:
        was_it_the_worst = "yes"
    else:
        was_it_the_worst = "no"

    if reflected_score < best_score:
        expansion_parameters = calculate_expansion(N, reflected_parameters, centroid_list, gamma)
        expansion_score = eval_func(expansion_parameters)
        if expansion_score < reflected_score:
            parameter_matrix, score_list = replace_point(N, worst_point, expansion_parameters, expansion_score, parameter_matrix,
score_list)

            print "Expansion point kept"

```

```

        else:
            parameter_matrix, score_list = replace_point(N, worst_point, reflected_parameters, reflected_score, parameter_matrix,
score_list)

            print "Reflected point kept"

        else:
            contraction_parameters = calculate_contracted(N, reflected_parameters, worst_list, centroid_list, was_it_the_worst, beta)
            contraction_score = eval_func(contraction_parameters)
            if contraction_score < worst_score:
                parameter_matrix, score_list = replace_point(N, worst_point, contraction_parameters, contraction_score, parameter_matrix,
score_list)

                print "Contraction point kept"

            else:
                parameter_matrix, score_list = move_incrementally(N, parameter_matrix, score_list)
                print "Moved incrementally"

            current_iter = current_iter + 1
            stop = check_for_stop(N, score_list)
            if current_iter >= 100:
                if int(score_array[current_iter - 100]) == int(best_score):
                    stop = "yes"

            best_score, best_score_index = find_best_score(N, score_list)
            return best_score, parameter_matrix[best_score_index]

def evaluate_scores(score, current_parameter_list, score_list, number_to_return):

    worst_score = score_list[0]
    worst_index = 0
    found_negative = "N"

    for lcv_eval in range(0, number_to_return):
        if (score_list[lcv_eval] < 0):
            worst_score = score_list[lcv_eval]
            worst_index = lcv_eval
            found_negative = "Y"
        elif (score_list[lcv_eval] > worst_score) and (found_negative == "N"):
            worst_score = score_list[lcv_eval]
            worst_index = lcv_eval

```

```
kept = "no"

if (worst_score > score) or (worst_score < 0):
    score_list[worst_index] = score
    genetic_parameters.starting_parameter_list[worst_index] = current_parameter_list
    kept = "yes"

return score_list, kept
```

```
def find_starting_points (number_to_return, number_of_tries, min_list, max_list):

    ofile = open(genetic_parameters.output_file, "w")

    output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"

    ofile.write(output_string)

    ofile.write(output_string)

    genetic_parameters.first_run = []

    for lcv in range (0, genetic_parameters.number_test_images):
        genetic_parameters.first_run.append("Y")

    images.raw_images = []
    images.lmin_images = []
    images.lmax_images = []
    images.x_dim = []
    images.y_dim = []
    images.lp_images = []
    images.edge_images = []
```

```

score_list = []

genetic_parameters.starting_parameter_list = []

genetic_parameters.verbose = "N"

for lcv_setup in range(0, number_to_return):
    parameters = []
    for lcv_param in range(0, genetic_parameters.number_of_parameters):
        parameters.append(1.1)
    score_list.append(-1)
    genetic_parameters.starting_parameter_list.append(parameters)

current_parameter_list = []

print "--> Generating starting points <--"

for lcv_try in range(0, number_of_tries):
    current_parameter_list = []
    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        value = min_list[lcv_parameter] + random.random() * (max_list[lcv_parameter] - min_list[lcv_parameter])
        current_parameter_list.append(value)
    score = eval_func(current_parameter_list)
    score_list, kept = evaluate_scores(score, current_parameter_list, score_list, number_to_return)
    print "Point", lcv_try, "Score =", score, "Kept =", kept
    if kept == "yes":
        print current_parameter_list

ofile.close()

def find_gain_offset():
    min_biomass = genetic_parameters.actual_biomass[0]
    max_biomass = genetic_parameters.actual_biomass[0]
    min_trees = genetic_parameters.actual_tree_count[0]
    max_trees = genetic_parameters.actual_tree_count[0]
    for lcv_list in range (1, genetic_parameters.number_test_images):

```

```

    if genetic_parameters.actual_biomass[lcv_list] < min_biomass:
        min_biomass = genetic_parameters.actual_biomass[lcv_list]
    if genetic_parameters.actual_tree_count[lcv_list] < min_trees:
        min_trees = genetic_parameters.actual_tree_count[lcv_list]
    if genetic_parameters.actual_biomass[lcv_list] > max_biomass:
        max_biomass = genetic_parameters.actual_biomass[lcv_list]
    if genetic_parameters.actual_tree_count[lcv_list] > max_trees:
        max_trees = genetic_parameters.actual_tree_count[lcv_list]
genetic_parameters.offset = min_biomass - min_trees
genetic_parameters.gain = (float(max_biomass) - float(min_biomass)) / (float(max_trees) - float(min_trees))
print "Offset:", genetic_parameters.offset, "Gain:", genetic_parameters.gain

```

```
def begin():
```

```
#
```

```
#This function will start the program out if it is being run as a stand alone program
```

```
#
```

```
    genetic_parameters.print_biomass_estimates = "Y"
```

```
genetic_parameters.output_file = "c:/temp/pca_3_results.txt"
```

```
    parameter_names = ["Alpha:", "Beta:", "Gamma:", "Center thresh.:", "Ratio thresh.:", "A. E. thresh.:", "Min. total:", "Decay:", "SD"]
```

```
genetic_parameters.number_test_images = 25
```

```
    genetic_parameters.image_names = []
```

```
    genetic_parameters.pixel_length = 0.5
```

```
    genetic_parameters.plot_radius = 30
```

```
    genetic_parameters.image_names.append("1_test.tif")
```

```
    genetic_parameters.image_names.append("2_test.tif")
```

```
    genetic_parameters.image_names.append("3_test.tif")
```

```
genetic_parameters.image_names.append("4_test.tif")
genetic_parameters.image_names.append("5_test.tif")
genetic_parameters.image_names.append("6_test.tif")
genetic_parameters.image_names.append("7_test.tif")
genetic_parameters.image_names.append("8_test.tif")
genetic_parameters.image_names.append("9_test.tif")
genetic_parameters.image_names.append("10_test.tif")
genetic_parameters.image_names.append("11_test.tif")
genetic_parameters.image_names.append("12_test.tif")
genetic_parameters.image_names.append("13_test.tif")
genetic_parameters.image_names.append("14_test.tif")
genetic_parameters.image_names.append("15_test.tif")
genetic_parameters.image_names.append("16_test.tif")
genetic_parameters.image_names.append("17_test.tif")
genetic_parameters.image_names.append("18_test.tif")
genetic_parameters.image_names.append("19_test.tif")
genetic_parameters.image_names.append("20_test.tif")
genetic_parameters.image_names.append("21_test.tif")
genetic_parameters.image_names.append("22_test.tif")
genetic_parameters.image_names.append("23_test.tif")
genetic_parameters.image_names.append("24_test.tif")
genetic_parameters.image_names.append("25_test.tif")
```

```
genetic_parameters.actual_biomass = []
```

```
genetic_parameters.actual_biomass.append(4583.629506)
genetic_parameters.actual_biomass.append(5858.515939)
genetic_parameters.actual_biomass.append(9609.94579)
genetic_parameters.actual_biomass.append(5501.271876)
genetic_parameters.actual_biomass.append(5260.806918)
genetic_parameters.actual_biomass.append(5305.864073)
genetic_parameters.actual_biomass.append(6168.282657)
genetic_parameters.actual_biomass.append(6572.09406)
genetic_parameters.actual_biomass.append(4318.75928)
genetic_parameters.actual_biomass.append(7275.570325)
genetic_parameters.actual_biomass.append(6508.293567)
```

```
genetic_parameters.actual_biomass.append(6161.366967)
genetic_parameters.actual_biomass.append(7450.332476)
genetic_parameters.actual_biomass.append(6107.840789)
genetic_parameters.actual_biomass.append(7410.48672)
genetic_parameters.actual_biomass.append(8393.972252)
genetic_parameters.actual_biomass.append(7957.680061)
genetic_parameters.actual_biomass.append(4961.825874)
genetic_parameters.actual_biomass.append(8112.538305)
genetic_parameters.actual_biomass.append(8139.150883)
genetic_parameters.actual_biomass.append(6798.087643)
genetic_parameters.actual_biomass.append(7947.414877)
genetic_parameters.actual_biomass.append(4749.384789)
genetic_parameters.actual_biomass.append(8020.826783)
genetic_parameters.actual_biomass.append(5190.049296)
```

```
genetic_parameters.actual_tree_count = []
```

```
genetic_parameters.actual_tree_count.append(79.87755)
genetic_parameters.actual_tree_count.append(80.5)
genetic_parameters.actual_tree_count.append(108.1818)
genetic_parameters.actual_tree_count.append(89.14286)
genetic_parameters.actual_tree_count.append(83.57143)
genetic_parameters.actual_tree_count.append(91.8169)
genetic_parameters.actual_tree_count.append(99.47826)
genetic_parameters.actual_tree_count.append(133.3333)
genetic_parameters.actual_tree_count.append(79.10448)
genetic_parameters.actual_tree_count.append(139.2157)
genetic_parameters.actual_tree_count.append(129.6364)
genetic_parameters.actual_tree_count.append(133.7333)
genetic_parameters.actual_tree_count.append(131.4667)
genetic_parameters.actual_tree_count.append(92.4)
genetic_parameters.actual_tree_count.append(114.6667)
genetic_parameters.actual_tree_count.append(99.66667)
genetic_parameters.actual_tree_count.append(108.9)
genetic_parameters.actual_tree_count.append(93.81356)
genetic_parameters.actual_tree_count.append(124.6667)
```

```
genetic_parameters.actual_tree_count.append(128.8)
genetic_parameters.actual_tree_count.append(94.05)
genetic_parameters.actual_tree_count.append(92.7)
genetic_parameters.actual_tree_count.append(99.75)
genetic_parameters.actual_tree_count.append(130.9)
genetic_parameters.actual_tree_count.append(91.66667)

find_gain_offset()

genetic_parameters.buffer = 10

genetic_parameters.relative_importance = 0.5

genetic_parameters.edge_method = "I"

genetic_parameters.filter = "L"

genetic_parameters.min_radius = 1

genetic_parameters.max_radius = 8

genetic_parameters.number_of_parameters = 9

genetic_parameters.counter = 0

genetic_parameters.initial_equation_parameters = [-0.69113, 3.31007, 0.85506]

number_to_return = 5

number_of_tries = 50

min_list = [0, -5, 0, 0, 2, 0, 0, -20, 1]
max_list = [5, 0, 5, 10, 12, 5, 10, 10, 2.5]

find_starting_points(number_to_return, number_of_tries, min_list, max_list)
```



```

for lcv_loop in range(0, number_to_return):

    print "-----"
    print "Trying starting point #", lcv_loop

    genetic_parameters.initial_parameter_list = genetic_parameters.starting_parameter_list[lcv_loop]

    print "Start parameters:", genetic_parameters.initial_parameter_list

final_score, final_parameter_list = nelder_main()

    print "Best parameters for this iteration ="

    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        print parameter_names[lcv_parameter], final_parameter_list[lcv_parameter]

    print "Evaluations:", genetic_parameters.counter

    print final_score

    genetic_parameters.initial_parameter_list = final_parameter_list

    converged = "no"

    old_score = int(final_score)

    print "Best equation parameters =", genetic_parameters.current_best_equation_parameters

    while converged == "no":
        print "Attempting a restart at the convergence point"
        final_score, final_parameter_list = nelder_main()
        if lcv_loop == 0 or final_score < best_score:
            best_score = final_score
            best_parameter_list = final_parameter_list
        print "Final score = ", final_score
        for lcv_parameter in range(0, genetic_parameters.number_of_parameters):

```

```

        print parameter_names[lcv_parameter], final_parameter_list[lcv_parameter]
    print "Best parameters =", genetic_parameters.current_best_equation_parameters
    if old_score == int(final_score):
        converged = "yes"
    else:
        old_score = int(final_score)
        genetic_parameters.initial_parameter_list = final_parameter_list

print "-----"
print "Best score = ", best_score
for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
    print parameter_names[lcv_parameter], best_parameter_list[lcv_parameter]
print "Evaluations:", genetic_parameters.counter

if __name__ == "__main__":
    begin()

```

D.2 Module `nelder_biomass_score.py`

```

from Tkinter import *
import math
import Image
import sys
import string
import genetic_parameters
import images
import gc
import array

def eval_func (parameter_list):
    score = 0
    p1 = parameter_list[0]
    p2 = parameter_list[1]
    p3 = parameter_list[2]

```

```

for lcv_stand in range(0, genetic_parameters.number_test_images):
    biomass_sum = 0
    for lcv_tree in range(0, genetic_parameters.tree_count[lcv_stand]):
        term_1 = p1 * math.log(genetic_parameters.tree_density[lcv_stand])
        if genetic_parameters.tree_heights[lcv_stand][lcv_tree] > 0:
            term_2 = p2 * math.log(genetic_parameters.tree_heights[lcv_stand][lcv_tree])
        else:
            term_2 = 0
        try:
            tree_biomass = math.exp(term_1 + term_2 + p3)
        except:
            tree_biomass = 0
        biomass_sum = biomass_sum + tree_biomass
    biomass_error = biomass_sum - genetic_parameters.actual_biomass[lcv_stand]
    score = score + (biomass_error * biomass_error) / float(1000)
return score

```

```

def print_estimates(parameter_list):

```

```

    p1 = parameter_list[0]
    p2 = parameter_list[1]
    p3 = parameter_list[2]

```

```

    print "======"

```

```

    for lcv_stand in range(0, genetic_parameters.number_test_images):
        biomass_sum = 0
        for lcv_tree in range(0, genetic_parameters.tree_count[lcv_stand]):
            term_1 = p1 * math.log(genetic_parameters.tree_density[lcv_stand])
            if genetic_parameters.tree_heights[lcv_stand][lcv_tree] > 0:
                term_2 = p2 * math.log(genetic_parameters.tree_heights[lcv_stand][lcv_tree])
            else:
                term_2 = 0
            tree_biomass = math.exp(term_1 + term_2 + p3)
            biomass_sum = biomass_sum + tree_biomass
        print "Stand: ", lcv_stand, " = ", biomass_sum

```

```

def print_scores(N, best_score, best_score_index, parameter_matrix):

#
#This function prints out the scores, origin and parameters of the extant organisms
#
    parameter_names = ["Parameter 1:", "Parameter 2:", "Parameter 3", "Parameter 4"]
    print "Information for organism: ", best_score_index
    print "Parameters:"
    for lcv_parameter in range(0, N):
        print parameter_names[lcv_parameter], parameter_matrix[best_score_index][lcv_parameter]
        print "-----"

def create_initial_values(N, a, initial_parameter_list):

    sqrt2 = math.pow(2, 0.5)
    sqrtNp1 = math.pow(N+1, 0.5)

    p = (a / (N * sqrt2)) * (sqrtNp1 + N - 1)
    q = (a / (N * sqrt2)) * (sqrtNp1 - 1)

    parameter_matrix = []

    parameter_matrix.append(initial_parameter_list)

    for lcv_y in range (1, N+1):
        parameter_row = array.array("f", [])
        for lcv_x in range (0, N):
            if lcv_y == lcv_x + 1:
                value_to_add = initial_parameter_list[lcv_x] + p
            else:
                value_to_add = initial_parameter_list[lcv_x] + q
            parameter_row.append(value_to_add)
        parameter_matrix.append(parameter_row)

    score_list = []

```

```

for lcv_y in range(0, N+1):
    score = eval_func(parameter_matrix[lcv_y])
    score_list.append(score)

return parameter_matrix, score_list

```

```

def calculate_reflected(N, parameter_matrix, score_list, alpha):

```

```

#
# Find the parameter value and number of the worst point
#

```

```

worst_score = score_list[0]
worst_point = 0

```

```

#
# Find the worst score and the number of the point with the worst score
#

```

```

for lcv in range(1, N+1):
    if score_list[lcv] > worst_score:
        worst_point = lcv
        worst_score = score_list[lcv]

```

```

#
# Find the centroids of all but the worst point
#

```

```

centroid_list = []

```

```

for lcv_x in range(0, N):
    parameter_sum = 0
    for lcv_y in range(0, N+1):
        parameter_sum = parameter_sum + parameter_matrix[lcv_y][lcv_x]
    centroid = (float(1) / float(N)) * (parameter_sum - parameter_matrix[worst_point][lcv_x])

```

```

        centroid_list.append(centroid)

#
# Find the parameters for the reflected point
#

reflected_parameters = []
worst_list = []

for lcv in range(0, N):
    reflected_value = centroid_list[lcv] + alpha * (centroid_list[lcv] - parameter_matrix[worst_point][lcv])
    reflected_parameters.append(reflected_value)
    worst_list.append(parameter_matrix[worst_point][lcv])

return reflected_parameters, centroid_list, worst_list, worst_point, worst_score

def calculate_contracted(N, reflected_parameters, worst_parameters, centroid_list, was_it_the_worst, beta):

#
# Calculates the value of a contracted point
#

contracted_parameters = []

if was_it_the_worst == "yes":
    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - worst_parameters[lcv])
        contracted_parameters.append(new_parameter_value)
else:
    for lcv in range(0, N):
        new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - reflected_parameters[lcv])
        contracted_parameters.append(new_parameter_value)

return contracted_parameters

```

```
def calculate_expansion(N, reflected_parameters, centroid_list, gamma):
```

```
    #  
    # Calculates the value of an expansion point  
    #  
    expansion_parameters = []  
  
    for lcv in range(0, N):  
        new_parameter_value = centroid_list[lcv] + gamma * (reflected_parameters[lcv] - centroid_list[lcv])  
        expansion_parameters.append(new_parameter_value)  
  
    return expansion_parameters
```

```
def move_incrementally(N, parameter_matrix, score_list):
```

```
    #  
    # Moves the points in the parameter matrix incrementally closer to the parameters of the best member of the population  
    #  
    best_score = score_list[0]  
    best_score_number = 0  
  
    for lcv in range(1, N+1):  
        if score_list[lcv] < best_score:  
            best_score = score_list[lcv]  
            best_score_number = lcv  
  
    for lcv_x in range(0, N):  
        for lcv_y in range(0, N+1):  
            moved_point = (parameter_matrix[lcv_y][lcv_x] + parameter_matrix[best_score_number][lcv_x]) / float(2)  
            parameter_matrix[lcv_y][lcv_x] = moved_point  
score_list = array.array('f', [])  
    for lcv in range(0, N+1):  
        score = eval_func(parameter_matrix[lcv])  
        score_list.append(score)
```

```

    return parameter_matrix, score_list

def find_best_score(N, score_list):
    best_score = score_list[0]
    best_score_index = 0
    for lcv in range (1, N+1):
        if score_list[lcv] < best_score:
            best_score = score_list[lcv]
            best_score_index = lcv

    return best_score, best_score_index

def replace_point(N, worst_point, best_parameters, best_score, parameter_matrix, score_list):
    parameter_matrix[worst_point] = best_parameters
    score_list[worst_point] = best_score
    return parameter_matrix, score_list

def nelder_main(initial_parameter_list):

    #
    # The main engine for the algorithm
    #

    N = 3
    a = 10
    alpha = 1.0
    beta = 0.5
    gamma = 2.0
    acc = 0.0001

    if genetic_parameters.verbose == "Y":
        print "Here in NBS..."
        print "Initial_parameter_list", initial_parameter_list
        print "Tree density", genetic_parameters.tree_density
    max_iter = 300

```



```

stop = "no"
current_iter = 0

parameter_matrix, score_list = create_initial_values(N, a, initial_parameter_list)

while (current_iter <= max_iter) and (stop == "no"):
    reflected_parameters, centroid_list, worst_list, worst_point, worst_score = calculate_reflected(N, parameter_matrix, score_list, alpha)
    reflected_score = eval_func(reflected_parameters)
    best_score, best_score_index = find_best_score(N, score_list)

    if reflected_score > worst_score:
        was_it_the_worst = "yes"
    else:
        was_it_the_worst = "no"

    if reflected_score < best_score:
        expansion_parameters = calculate_expansion(N, reflected_parameters, centroid_list, gamma)
        expansion_score = eval_func(expansion_parameters)
        if expansion_score < reflected_score:
            parameter_matrix, score_list = replace_point(N, worst_point, expansion_parameters, expansion_score, parameter_matrix,
score_list)

        else:
            parameter_matrix, score_list = replace_point(N, worst_point, reflected_parameters, reflected_score, parameter_matrix,
score_list)

    else:
        contraction_parameters = calculate_contracted(N, reflected_parameters, worst_list, centroid_list, was_it_the_worst, beta)
        contraction_score = eval_func(contraction_parameters)
        if contraction_score < worst_score:
            parameter_matrix, score_list = replace_point(N, worst_point, contraction_parameters, contraction_score, parameter_matrix,
score_list)

        else:
            parameter_matrix, score_list = move_incrementally(N, parameter_matrix, score_list)

```

```
        current_iter = current_iter + 1

genetic_parameters.current_best_equation_parameters = parameter_matrix[best_score_index]

#This corrects an odd bug in Python...

if genetic_parameters.current_best_equation_parameters == [0,0]:
    pass

if genetic_parameters.print_biomass_estimates == "Y":
    print_estimates(parameter_matrix[best_score_index])
    print_scores(N, score_list[best_score_index], best_score_index, parameter_matrix)
return best_score
```

Appendix E. The program for setting parameters for the program that locates trees in LiDAR images and does not simultaneously determines biomass.

```
from Tkinter import *
import math
import random
import Image
import sys
import string
import genetic_parameters
import images
import gc
import array

def raw(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    x_dim, y_dim = in_image.size
    images.x_dim.append(x_dim)
    images.y_dim.append(y_dim)
    images.raw_images.append(in_image)

def edge(image_number):
    edge_raw = Image.open(genetic_parameters.image_names[image_number])
    return edge_raw

def low_pass(source_image, image_number):
    lp_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    si_array = array.array("d")
    si_array.fromlist(list(source_image.getdata()))
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            if lcv_row == 0 or lcv_row == images.y_dim[image_number] - 1 or lcv_column == 0 or lcv_column ==
images.x_dim[image_number] - 1:
                value = si_array[lcv_row * x_dim + lcv_column]
```

```

        lp_image.putpixel((lcv_column, lcv_row), value)
else:
    value = si_array[lcv_row * x_dim + lcv_column]
    value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column - 1)]
    value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column)]
    value = value + si_array[((lcv_row - 1) * x_dim + (lcv_column + 1))]
    value = value + si_array[((lcv_row) * x_dim + (lcv_column - 1))]
    value = value + si_array[((lcv_row) * x_dim + (lcv_column + 1))]
    value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column - 1))]
    value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column))]
    value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column + 1))]
    value = float(value)/float(9)
    lp_image.putpixel((lcv_column, lcv_row), value)

return lp_image

def calculate_dog(SD):
    multipliers = []
    pi = math.pi
    sqrt2 = math.sqrt(2)
    for lcv_x in range(-5, 6):
        lcv_calculate = float(lcv_x)
        term_1 = -0.5 * sqrt2 / (math.sqrt(pi) * math.pow(SD, 3))
        term_2 = math.exp(-0.5 * ((lcv_calculate * lcv_calculate) / (SD * SD)))
        term_3 = 0.5 * math.sqrt(2) / (math.sqrt(pi) * math.pow(SD, 5))
        term_4 = (lcv_calculate) * (lcv_calculate)
        term_5 = math.exp(-0.5 * (lcv_calculate * lcv_calculate) / (SD * SD))
        coef = (term_1 * term_2) + (term_3 * term_4 * term_5)
        multipliers.append(coef)

    return multipliers

def lmin(edge_raw, image_number, SD):
    lm_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)

```

```

lmax_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
x_dim = images.x_dim[image_number]
y_dim = images.y_dim[image_number]
multipliers = calculate_dog(SD)
edge_array = array.array("f")
edge_array.fromlist(list(edge_raw.getdata()))

pp_lm = lm_image.putpixel
pp_lmax = lmax_image.putpixel

for lcv_row in range (6, images.y_dim[image_number] - 6):
    for lcv_column in range (6, images.x_dim[image_number] - 6):
        value_01 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[lcv_column-1 + (lcv_row + 1) * x_dim] +
edge_array[lcv_column+1 + (lcv_row - 1) * x_dim]
        value_1 = edge_array[(lcv_column-5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column-6) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row - 6) * x_dim]
        value_2 = edge_array[(lcv_column-4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column-5) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row - 5) * x_dim]
        value_3 = edge_array[(lcv_column-3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column-4) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row - 4) * x_dim]
        value_4 = edge_array[(lcv_column-2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column-3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row - 3) * x_dim]
        value_5 = edge_array[(lcv_column-1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column-2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row - 2) * x_dim]
        value_6 = edge_array[(lcv_column+1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
        value_7 = edge_array[(lcv_column+2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
        value_8 = edge_array[(lcv_column+3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 2) * x_dim]
        value_9 = edge_array[(lcv_column+4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 3) * x_dim]
        value_10 = edge_array[(lcv_column+5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row + 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row + 4) * x_dim]
        value_11 = edge_array[(lcv_column) + (lcv_row-5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 5) * x_dim]

```

```

        value_12 = edge_array[(lcv_column) + (lcv_row-4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 4) * x_dim]
        value_13 = edge_array[(lcv_column) + (lcv_row-3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 3) * x_dim]
        value_14 = edge_array[(lcv_column) + (lcv_row-2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 2) * x_dim]
        value_15 = edge_array[(lcv_column) + (lcv_row-1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 1) * x_dim]
        value_02 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row) * x_dim]
        value_16 = edge_array[(lcv_column) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
        value_17 = edge_array[(lcv_column) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 2) * x_dim]
        value_18 = edge_array[(lcv_column) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 3) * x_dim]
        value_19 = edge_array[(lcv_column) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 4) * x_dim]
        value_20 = edge_array[(lcv_column) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 5) * x_dim]
        value_21 = edge_array[(lcv_column-5) + (lcv_row) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 5 + (lcv_row + 1) * x_dim]
        value_22 = edge_array[(lcv_column-4) + (lcv_row) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 1) * x_dim]
        value_23 = edge_array[(lcv_column-3) + (lcv_row) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 1) * x_dim]
        value_24 = edge_array[(lcv_column-2) + (lcv_row) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 1) * x_dim]
        value_25 = edge_array[(lcv_column-1) + (lcv_row) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 1) * x_dim]
        value_03 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + (lcv_row + 1) * x_dim]
        value_26 = edge_array[(lcv_column+1) + (lcv_row) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
        value_27 = edge_array[(lcv_column+2) + (lcv_row) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row + 1) * x_dim]

```

```

value_28 = edge_array[(lcv_column+3) + (lcv_row) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
value_29 = edge_array[(lcv_column+4) + (lcv_row) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 1) * x_dim]
value_30 = edge_array[(lcv_column+5) + (lcv_row) * x_dim] + edge_array[(lcv_column + 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 1) * x_dim]
value_31 = edge_array[(lcv_column-5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 6) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 6) * x_dim]
value_32 = edge_array[(lcv_column-4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 5) * x_dim]
value_33 = edge_array[(lcv_column-3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 4) * x_dim]
value_34 = edge_array[(lcv_column-2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 3) * x_dim]
value_35 = edge_array[(lcv_column-1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row + 2) * x_dim]
value_04 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
value_36 = edge_array[(lcv_column+1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
value_37 = edge_array[(lcv_column+2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row - 1) * x_dim]
value_38 = edge_array[(lcv_column+3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row - 2) * x_dim]
value_39 = edge_array[(lcv_column+4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row - 3) * x_dim]
value_40 = edge_array[(lcv_column+5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row - 4) * x_dim]

```

```
lm_list = []
```

```

lm_1 = multipliers[0]*value_1 + multipliers[1]*value_2 + multipliers[2]*value_3 + multipliers[3]*value_4 + multipliers[4] *
value_5 + multipliers[5] * value_01 + multipliers[6] * value_6 + multipliers[7]*value_7 + multipliers[8]*value_8 + multipliers[9] * value_9 + multipliers[10]
* value_10

```

```

lm_2 = multipliers[0]*value_11 + multipliers[1]*value_12 + multipliers[2]*value_13 + multipliers[3]*value_14 + multipliers[4] *
value_15 + multipliers[5] * value_02 + multipliers[6] * value_16 + multipliers[7]*value_17 + multipliers[8]*value_18 + multipliers[9] * value_19 +

```

```

multipliers[10] * value_20
    lm_3 = multipliers[0]*value_21 + multipliers[1]*value_22 + multipliers[2]*value_23 + multipliers[3]*value_24 + multipliers[4] *
value_25 + multipliers[5] * value_03 + multipliers[6] * value_26 + multipliers[7]*value_27 + multipliers[8]*value_28 + multipliers[9] * value_29 +
multipliers[10] * value_30
    lm_4 = multipliers[0]*value_31 + multipliers[1]*value_32 + multipliers[2]*value_33 + multipliers[3]*value_34 + multipliers[4] *
value_35 + multipliers[5] * value_04 + multipliers[6] * value_36 + multipliers[7]*value_37 + multipliers[8]*value_38 + multipliers[9] * value_39 +
multipliers[10] * value_40
    lm_list.append(lm_1)
    lm_list.append(lm_2)
    lm_list.append(lm_3)
    lm_list.append(lm_4)
    lm_list.sort()
    pp_lm((lcv_column, lcv_row), lm_list[3])
    pp_lmax((lcv_column, lcv_row), lm_list[0])
return lm_image, lmax_image

```

```

def make_int_maps(image_number):
    images.value_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    images.radius_image = Image.new("L", (images.x_dim[image_number], images.y_dim[image_number]), 0)

```

```

def find_value_diam(image_number, center_thresh, ratio_thresh, ae_thresh, min_total, decay):

```

```

    if min_total < 0:
        min_total = 0

    edge_array = array.array("f")
    edge_array.fromlist(list(images.edge_images[image_number].getdata()))

    lmin_array = array.array("f")
    lmin_array.fromlist(list(images.lmin_image.getdata()))

    raw_array = array.array("f")
    raw_array.fromlist(list(images.raw_images[image_number].getdata()))

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

```



```

pp_value = images.value_image.putpixel
pp_radius = images.radius_image.putpixel

for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        test_1 = lmin_array[lcv_line * x_dim + lcv_column]
        test_2 = raw_array[lcv_line * x_dim + lcv_column]
        if test_1 < center_thresh and test_2 > ratio_thresh:
            high_value = -999
            high_radius = 0
            stop = "N"
            stop_now = "N"
            for lcv_radius in range(genetic_parameters.min_radius, genetic_parameters.max_radius + 1):
                if stop == "Y":
                    stop_now = "Y"
                pixels = 0
                value = 0
                sqrad = lcv_radius * lcv_radius
                for lcv_y in range (-1 * lcv_radius, lcv_radius + 1):
                    x_radius = int(round(math.sqrt(sqrad - lcv_y * lcv_y)))
                    position_y = lcv_line + lcv_y
                    pos_position_x = lcv_column + x_radius
                    neg_position_x = lcv_column - x_radius
                    if (test_2 - edge_array[position_y * x_dim + pos_position_x]) > ae_thresh:
                        stop = "Y"
                    if (test_2 - edge_array[position_y * x_dim + neg_position_x]) > ae_thresh:
                        stop = "Y"
                    value = value + lmin_array[position_y * x_dim + pos_position_x]
                    value = value + lmin_array[position_y * x_dim + neg_position_x]
                    if x_radius == 0:
                        pixels = pixels + 1
                    else:
                        pixels = pixels + 2

            total_value = float(value) / float(pixels)
            total_value = total_value + decay*lcv_radius

```

```

    if total_value > high_value:
        high_value = total_value
        high_radius = lcv_radius
    if high_value < min_total:
        high_value = 0
    if stop_now == "Y":
        break

```

```

    pp_value((lcv_column, lcv_line), high_value)
    pp_radius((lcv_column, lcv_line), high_radius)

```

```

def refine_trees(image_number, alpha, beta, gamma):

```

```

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

```

```

    lmax_array = array.array("f")
    lmax_array.fromlist(list(images.lmax_image.getdata()))

```

```

    radius_array = array.array("f")
    radius_array.fromlist(list(images.radius_image.getdata()))

```

```

    value_array = array.array("f")
    value_array.fromlist(list(images.value_image.getdata()))

```

```

    raw_array = array.array("f")
    raw_array.fromlist(list(images.raw_images[image_number].getdata()))

```

```

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
            if pixel_rad > 0:
                keep = 1
                score = alpha * value_array[lcv_line * x_dim + lcv_column]
                score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]

```

```

score = score + gamma * raw_array[lcv_line * x_dim + lcv_column]
for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
    for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
        radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
        ignore = "N"
        if round(radius) > pixel_rad:
            ignore = "Y"
        if ignore == "N":
            wscore = alpha * value_array[win_y * x_dim + win_x]
            wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
            wscore = wscore + gamma * raw_array[win_y * x_dim + win_x]
            if wscore > score:
                keep = 0
                break
    if int(keep) == 0:
        break
images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

```

```

radius_array = array.array("f")
radius_array.fromlist(list(images.radius_image.getdata()))

```

#The following routine cleans up any small trees that are encroaching on a large tree's space

```

for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_array[lcv_line * x_dim + lcv_column]
            score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
            score = score + gamma * raw_array[lcv_line * x_dim + lcv_column]
            for win_y in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                    radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                    ignore = "N"
                    if round(radius) > pixel_rad:

```

```

        ignore = "Y"
    if ignore == "N" and radius_array[win_y * x_dim + win_x] > 0:
        wscore = alpha * value_array[win_y * x_dim + win_x]
        wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
        wscore = wscore + gamma * raw_array[win_y * x_dim + win_x]
        if wscore < score:
            images.radius_image.putpixel((win_x, win_y), 0)

```

```

def find_height(lcv_column, lcv_line, pixel_rad, image_number):
    max_height = -2000
    for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
        for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
            radius = math.sqrt((win_x - lcv_column) * (win_x - lcv_column) + (win_y - lcv_line) * (win_y - lcv_line))
            if round(radius) <= pixel_rad:
                lidar_value = images.raw_images[image_number].getpixel((win_x, win_y))
                if lidar_value > max_height:
                    max_height = lidar_value

    if max_height == -2000:
        print pixel_rad
    return max_height

```

```

def make_output(image_number):
    trees = 0
    diameter_list = []
    height_list = []
    for lcv_line in range(genetic_parameters.buffer, images.y_dim[image_number] - genetic_parameters.buffer):
        for lcv_column in range(genetic_parameters.buffer, images.x_dim[image_number] - genetic_parameters.buffer):
            pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))
            center_x = float(images.x_dim[image_number]) / float(2)
            center_y = float(images.y_dim[image_number]) / float(2)
            term_1 = (center_x - float(lcv_column)) * (center_x - float(lcv_column))
            term_2 = (center_y - float(lcv_line)) * (center_y - float(lcv_line))
            distance = math.sqrt(term_1 + term_2)
            if pixel_rad > 0 and int(round(distance)) <= genetic_parameters.plot_radius:

```

```

        trees = trees + 1
        diameter = genetic_parameters.pixel_length + 2 * genetic_parameters.pixel_length * pixel_rad
        diameter_list.append(diameter)
        height = find_height(lcv_column, lcv_line, pixel_rad, image_number)
        height_list.append(height)
area = (math.pi * math.pow(genetic_parameters.plot_radius * genetic_parameters.pixel_length, 2)) / 10000
density = trees / area
genetic_parameters.tree_count.append(trees)
genetic_parameters.tree_density.append(density)
genetic_parameters.crown_widths.append(diameter_list)
genetic_parameters.tree_heights.append(height_list)

```

```
def start(image_number, alpha, beta, gamma, center_thresh, ratio_thresh, ae_thresh, min_total, decay, SD):
```

```

    genetic_parameters.counter = genetic_parameters.counter + 1
    if genetic_parameters.first_run[image_number] == "Y":
        raw(image_number)
    if genetic_parameters.first_run[image_number] == "Y":
        images.lp_images.append(low_pass(images.raw_images[image_number], image_number))
    if genetic_parameters.first_run[image_number] == "Y":
        if genetic_parameters.edge_method == "R":
            images.edge_images.append(lp_image)
        else:
            edge_raw = edge(image_number)
            images.edge_images.append(low_pass(edge_raw, image_number))
    else:
        edge_raw = edge(image_number)
    images.lmin_image, images.lmax_image = lmin(edge_raw, image_number, SD)
    make_int_maps(image_number)
    find_value_diam(image_number, center_thresh, ratio_thresh, ae_thresh, min_total, decay)
    refine_trees(image_number, alpha, beta, gamma)
    make_output(image_number)
    del images.radius_image
    del images.value_image

```

```
#
```

```
# Nelder and Meade Simplex Algorithm
#
```

```
def eval_func (parameter_list):
```

```
    genetic_parameters.crown_widths = []
    genetic_parameters.tree_density = []
    genetic_parameters.tree_count = []
    genetic_parameters.tree_heights = []
```

```
    for lcv_image in range (0, genetic_parameters.number_test_images):
```

```
        alpha = parameter_list[0]
```

```
        beta = parameter_list[1]
```

```
        gamma = parameter_list[2]
```

```
        center = parameter_list[3]
```

```
        ratio = parameter_list[4]
```

```
        ae = parameter_list[5]
```

```
        min = parameter_list[6]
```

```
        decay = parameter_list[7]
```

```
        SD = parameter_list[8]
```

```
        start(lcv_image, alpha, beta, gamma, center, ratio, ae, min, decay, SD)
```

```
    genetic_parameters.first_run[lcv_image] = "N"
```

```
    genetic_parameters.print_biomass_estimates = "N"
```

```
    tree_score = 0
```

```
    for lcv_image in range (0, genetic_parameters.number_test_images):
```

```
        tree_count_error = genetic_parameters.actual_tree_count[lcv_image] - genetic_parameters.tree_count[lcv_image]
```

```
        tree_score = tree_score + tree_count_error * tree_count_error
```

```
    score = tree_score
```

```
    return score
```

```

def create_initial_values(N, a, initial_parameter_list):

    sqrt2 = math.pow(2, 0.5)
    sqrtNp1 = math.pow(N+1, 0.5)

    parameter_matrix = []

    parameter_matrix.append(initial_parameter_list)

    for lcv_y in range(1, N+1):
        parameter_row = array.array("f", [])
        for lcv_x in range(0, N):
            p = (a[lcv_x] / (N * sqrt2)) * (sqrtNp1 + N - 1)
            q = (a[lcv_x] / (N * sqrt2)) * (sqrtNp1 - 1)
            if lcv_y == lcv_x + 1:
                value_to_add = initial_parameter_list[lcv_x] + p
            else:
                value_to_add = initial_parameter_list[lcv_x] + q
            parameter_row.append(value_to_add)
        parameter_matrix.append(parameter_row)

    #score_list = array.array("f")
    score_list = []

    for lcv_y in range(0, N+1):
        score = eval_func(parameter_matrix[lcv_y])
        score_list.append(score)

    print "Score for point 0:", score_list[0]
    print "Parameters for point 0:", parameter_matrix[0]

    return parameter_matrix, score_list

def calculate_reflected(N, parameter_matrix, score_list, alpha):

```

```

#
# Find the parameter value and number of the worst point
#

worst_score = score_list[0]
worst_point = 0

#
# Find the worst score and the number of the point with the worst score
#

for lcv in range(1, N+1):
    if score_list[lcv] > worst_score:
        worst_point = lcv
        worst_score = score_list[lcv]

#
# Find the centroids of all but the worst point
#

centroid_list = []

for lcv_x in range(0, N):
    parameter_sum = 0
    for lcv_y in range(0, N+1):
        parameter_sum = parameter_sum + parameter_matrix[lcv_y][lcv_x]
    centroid = (float(1) / float(N))* (parameter_sum - parameter_matrix[worst_point][lcv_x])
    centroid_list.append(centroid)

#
# Find the parameters for the reflected point
#

reflected_parameters = []
worst_list = []

```



```

for lcv in range(0, N):
    reflected_value = centroid_list[lcv] + alpha * (centroid_list[lcv] - parameter_matrix[worst_point][lcv])
    reflected_parameters.append(reflected_value)
    worst_list.append(parameter_matrix[worst_point][lcv])

return reflected_parameters, centroid_list, worst_list, worst_point, worst_score

```

```

def calculate_contracted(N, reflected_parameters, worst_parameters, centroid_list, was_it_the_worst, beta):

```

```

    #
    # Calculates the value of a contracted point
    #

    contracted_parameters = []

    if was_it_the_worst == "yes":
        for lcv in range(0, N):
            new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - worst_parameters[lcv])
            contracted_parameters.append(new_parameter_value)
    else:
        for lcv in range(0, N):
            new_parameter_value = centroid_list[lcv] - beta*(centroid_list[lcv] - reflected_parameters[lcv])
            contracted_parameters.append(new_parameter_value)

    return contracted_parameters

```

```

def calculate_expansion(N, reflected_parameters, centroid_list, gamma):

```

```

    #
    # Calculates the value of an expansion point
    #

    expansion_parameters = []

    for lcv in range(0, N):

```

```
new_parameter_value = centroid_list[lcv] + gamma * (reflected_parameters[lcv] - centroid_list[lcv])
expansion_parameters.append(new_parameter_value)
```

```
return expansion_parameters
```

```
def move_incrementally(N, parameter_matrix, score_list):
```

```
#
# Moves the points in the parameter matrix incrementally closer to the parameters of the best member of the population
#
```

```
best_score = score_list[0]
best_score_number = 0
```

```
for lcv in range(1, N+1):
    if score_list[lcv] < best_score:
        best_score = score_list[lcv]
        best_score_number = lcv
```

```
for lcv_x in range(0, N):
    for lcv_y in range(0, N+1):
        moved_point = (parameter_matrix[lcv_y][lcv_x] + parameter_matrix[best_score_number][lcv_x]) / float(2)
        parameter_matrix[lcv_y][lcv_x] = moved_point
score_list = array.array('f', [])
for lcv in range(0, N+1):
    score = eval_func(parameter_matrix[lcv])
    score_list.append(score)
```

```
return parameter_matrix, score_list
```

```
def find_best_score(N, score_list):
    print "Score 0 = ", score_list[0]
    best_score = score_list[0]
    best_score_index = 0
```

```

for lcv in range (1, N+1):
    if score_list[lcv] < best_score:
        best_score = score_list[lcv]
        best_score_index = lcv

return best_score, best_score_index

def check_for_stop(N, score_list):
    first_score = int(10 * score_list[0])
    stop = "yes"
    for lcv in range(1, N+1):
        if first_score <> int(10 * score_list[lcv]):
            stop = "no"
    return stop

def replace_point(N, worst_point, best_parameters, best_score, parameter_matrix, score_list):
    parameter_matrix[worst_point] = best_parameters
    score_list[worst_point] = best_score
    return parameter_matrix, score_list

def nelder_main():

    #
    # The main engine for the algorithm
    #

    genetic_parameters.verbose = "N"

    print "Up here..."

    ofile = open(genetic_parameters.output_file, "w")

    output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"

    ofile.write(output_string)

```

```

ofile.write(output_string)

images.raw_images = []
images.lmin_images = []
images.lmax_images = []
images.x_dim = []
images.y_dim = []
images.lp_images = []
images.edge_images = []

genetic_parameters.best_equation_parameter_list = []

genetic_parameters.first_run = []

for lcv in range (0, genetic_parameters.number_test_images):
    genetic_parameters.first_run.append("Y")

N = genetic_parameters.number_of_parameters
a = [5,5,5,10,10,5,10,30,1.5]
alpha = 1.0
beta = 0.5
gamma = 2.0
acc = 0.0001
initial_parameter_list = genetic_parameters.initial_parameter_list
score = eval_func(initial_parameter_list)

max_iter = 1000
stop = "no"
current_iter = 0

parameter_matrix, score_list = create_initial_values(N, a, initial_parameter_list)

score_array = []

while (current_iter <= max_iter) and (stop == "no"):
    reflected_parameters, centroid_list, worst_list, worst_point, worst_score = calculate_reflected(N, parameter_matrix, score_list, alpha)

```

```

reflected_score = eval_func(reflected_parameters)
best_score, best_score_index = find_best_score(N, score_list)
score_array.append(best_score)
print "Best score = ", best_score
print "Best score index = ", best_score_index
print "Current iteration = ", current_iter
print "-----"

if reflected_score > worst_score:
    was_it_the_worst = "yes"
else:
    was_it_the_worst = "no"

if reflected_score < best_score:
    expansion_parameters = calculate_expansion(N, reflected_parameters, centroid_list, gamma)
    expansion_score = eval_func(expansion_parameters)
    if expansion_score < reflected_score:
        parameter_matrix, score_list = replace_point(N, worst_point, expansion_parameters, expansion_score, parameter_matrix,
score_list)
        print "Expansion point kept"
    else:
        parameter_matrix, score_list = replace_point(N, worst_point, reflected_parameters, reflected_score, parameter_matrix,
score_list)
        print "Reflected point kept"
else:
    contraction_parameters = calculate_contracted(N, reflected_parameters, worst_list, centroid_list, was_it_the_worst, beta)
    contraction_score = eval_func(contraction_parameters)
    if contraction_score < worst_score:
        parameter_matrix, score_list = replace_point(N, worst_point, contraction_parameters, contraction_score, parameter_matrix,
score_list)
        print "Contraction point kept"
    else:
        parameter_matrix, score_list = move_incrementally(N, parameter_matrix, score_list)
        print "Moved incrementally"
current_iter = current_iter + 1
stop = check_for_stop(N, score_list)

```

```

        if current_iter >= 100:
            if int(score_array[current_iter - 100]) == int(best_score):
                stop = "yes"
    best_score, best_score_index = find_best_score(N, score_list)
    return best_score, parameter_matrix[best_score_index]

```

```

def evaluate_scores(score, current_parameter_list, score_list, number_to_return):

```

```

    worst_score = score_list[0]
    worst_index = 0
    found_negative = "N"

    for lcv_eval in range(0, number_to_return):
        if (score_list[lcv_eval] < 0):
            worst_score = score_list[lcv_eval]
            worst_index = lcv_eval
            found_negative = "Y"
        elif (score_list[lcv_eval] > worst_score) and (found_negative == "N"):
            worst_score = score_list[lcv_eval]
            worst_index = lcv_eval

    kept = "no"

    if (worst_score > score) or (worst_score < 0):
        score_list[worst_index] = score
        genetic_parameters.starting_parameter_list[worst_index] = current_parameter_list
        kept = "yes"

    return score_list, kept

```

```

def find_starting_points (number_to_return, number_of_tries, min_list, max_list):

```

```

    ofile = open(genetic_parameters.output_file, "w")

```

```

output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"

ofile.write(output_string)

ofile.write(output_string)

genetic_parameters.first_run = []

for lcv in range(0, genetic_parameters.number_test_images):
    genetic_parameters.first_run.append("Y")

images.raw_images = []
images.lmin_images = []
images.lmax_images = []
images.x_dim = []
images.y_dim = []
images.lp_images = []
images.edge_images = []

score_list = []

genetic_parameters.starting_parameter_list = []

genetic_parameters.verbose = "N"

for lcv_setup in range(0, number_to_return):
    parameters = []
    for lcv_param in range(0, genetic_parameters.number_of_parameters):
        parameters.append(1.1)
    score_list.append(-1)
    genetic_parameters.starting_parameter_list.append(parameters)

current_parameter_list = []

print "--> Generating starting points <--"

```

```

for lcv_try in range(0, number_of_tries):
    current_parameter_list = []
    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        value = min_list[lcv_parameter] + random.random() * (max_list[lcv_parameter] - min_list[lcv_parameter])
        current_parameter_list.append(value)
    score = eval_func(current_parameter_list)
    score_list, kept = evaluate_scores(score, current_parameter_list, score_list, number_to_return)
    print "Point", lcv_try, "Score =", score, "Kept =", kept
    if kept == "yes":
        print current_parameter_list

ofile.close()

def find_gain_offset():
    min_biomass = genetic_parameters.actual_biomass[0]
    max_biomass = genetic_parameters.actual_biomass[0]
    min_trees = genetic_parameters.actual_tree_count[0]
    max_trees = genetic_parameters.actual_tree_count[0]
    for lcv_list in range(1, genetic_parameters.number_test_images):
        if genetic_parameters.actual_biomass[lcv_list] < min_biomass:
            min_biomass = genetic_parameters.actual_biomass[lcv_list]
        if genetic_parameters.actual_tree_count[lcv_list] < min_trees:
            min_trees = genetic_parameters.actual_tree_count[lcv_list]
        if genetic_parameters.actual_biomass[lcv_list] > max_biomass:
            max_biomass = genetic_parameters.actual_biomass[lcv_list]
        if genetic_parameters.actual_tree_count[lcv_list] > max_trees:
            max_trees = genetic_parameters.actual_tree_count[lcv_list]
    genetic_parameters.offset = min_biomass - min_trees
    genetic_parameters.gain = (float(max_biomass) - float(min_biomass)) / (float(max_trees) - float(min_trees))
    print "Offset:", genetic_parameters.offset, "Gain:", genetic_parameters.gain

def begin():
    #
    #This function will start the program out if it is being run as a stand alone program

```



```
#
```

```
genetic_parameters.print_biomass_estimates = "Y"
```

```
genetic_parameters.output_file = "c:/temp/pca_3_results.txt"
```

```
parameter_names = ["Alpha:", "Beta:", "Gamma:", "Center thresh.:", "Ratio thresh.:", "A. E. thresh.:", "Min. total:", "Decay:", "SD"]
```

```
genetic_parameters.number_test_images = 25
```

```
genetic_parameters.image_names = []
```

```
genetic_parameters.pixel_length = 0.5
```

```
genetic_parameters.plot_radius = 30
```

```
genetic_parameters.image_names.append("1_test.tif")
```

```
genetic_parameters.image_names.append("2_test.tif")
```

```
genetic_parameters.image_names.append("3_test.tif")
```

```
genetic_parameters.image_names.append("4_test.tif")
```

```
genetic_parameters.image_names.append("5_test.tif")
```

```
genetic_parameters.image_names.append("6_test.tif")
```

```
genetic_parameters.image_names.append("7_test.tif")
```

```
genetic_parameters.image_names.append("8_test.tif")
```

```
genetic_parameters.image_names.append("9_test.tif")
```

```
genetic_parameters.image_names.append("10_test.tif")
```

```
genetic_parameters.image_names.append("11_test.tif")
```

```
genetic_parameters.image_names.append("12_test.tif")
```

```
genetic_parameters.image_names.append("13_test.tif")
```

```
genetic_parameters.image_names.append("14_test.tif")
```

```
genetic_parameters.image_names.append("15_test.tif")
```

```
genetic_parameters.image_names.append("16_test.tif")
```

```
genetic_parameters.image_names.append("17_test.tif")
```

```
genetic_parameters.image_names.append("18_test.tif")
```

```
genetic_parameters.image_names.append("19_test.tif")
```

```
genetic_parameters.image_names.append("20_test.tif")
```

```
genetic_parameters.image_names.append("21_test.tif")
genetic_parameters.image_names.append("22_test.tif")
genetic_parameters.image_names.append("23_test.tif")
genetic_parameters.image_names.append("24_test.tif")
genetic_parameters.image_names.append("25_test.tif")
```

```
genetic_parameters.actual_tree_count = []
```

```
genetic_parameters.actual_tree_count.append(103)
genetic_parameters.actual_tree_count.append(92)
genetic_parameters.actual_tree_count.append(119)
genetic_parameters.actual_tree_count.append(96)
genetic_parameters.actual_tree_count.append(90)
genetic_parameters.actual_tree_count.append(123)
genetic_parameters.actual_tree_count.append(104)
genetic_parameters.actual_tree_count.append(144)
genetic_parameters.actual_tree_count.append(106)
genetic_parameters.actual_tree_count.append(142)
genetic_parameters.actual_tree_count.append(138)
genetic_parameters.actual_tree_count.append(136)
genetic_parameters.actual_tree_count.append(136)
genetic_parameters.actual_tree_count.append(99)
genetic_parameters.actual_tree_count.append(160)
genetic_parameters.actual_tree_count.append(130)
genetic_parameters.actual_tree_count.append(121)
genetic_parameters.actual_tree_count.append(123)
genetic_parameters.actual_tree_count.append(136)
genetic_parameters.actual_tree_count.append(138)
genetic_parameters.actual_tree_count.append(99)
genetic_parameters.actual_tree_count.append(103)
genetic_parameters.actual_tree_count.append(105)
genetic_parameters.actual_tree_count.append(154)
genetic_parameters.actual_tree_count.append(110)
```

```
find_gain_offset()
```

```
genetic_parameters.buffer = 10

genetic_parameters.relative_importance = 0.5

genetic_parameters.edge_method = "I"

genetic_parameters.filter = "L"

genetic_parameters.min_radius = 1

genetic_parameters.max_radius = 8

genetic_parameters.number_of_parameters = 9

genetic_parameters.counter = 0

number_to_return = 5

number_of_tries = 50

min_list = [0, -5, 0, 0, 2, 0, 0, -20, 1]
max_list = [5, 0, 5, 10, 12, 5, 10, 10, 2.5]

find_starting_points(number_to_return, number_of_tries, min_list, max_list)

for lcv_loop in range(0, number_to_return):

    print "-----"
    print "Trying starting point #", lcv_loop

    genetic_parameters.initial_parameter_list = genetic_parameters.starting_parameter_list[lcv_loop]

    print "Start parameters:", genetic_parameters.initial_parameter_list

final_score, final_parameter_list = nelder_main()
```

```

print "Best parameters for this iteration ="

for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
    print parameter_names[lcv_parameter], final_parameter_list[lcv_parameter]

print "Evaluations:", genetic_parameters.counter

print final_score

genetic_parameters.initial_parameter_list = final_parameter_list

converged = "no"

old_score = int(final_score)

#print "Best equation parameters =", genetic_parameters.current_best_equation_parameters

while converged == "no":
    print "Attempting a restart at the convergence point"
    final_score, final_parameter_list = nelder_main()
    if lcv_loop == 0 or final_score < best_score:
        best_score = final_score
        best_parameter_list = final_parameter_list
    print "Final score = ", final_score
    for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
        print parameter_names[lcv_parameter], final_parameter_list[lcv_parameter]
    if old_score == int(final_score):
        converged = "yes"
    else:
        old_score = int(final_score)
        genetic_parameters.initial_parameter_list = final_parameter_list

print "-----"
print "Best score = ", best_score
for lcv_parameter in range(0, genetic_parameters.number_of_parameters):
    print parameter_names[lcv_parameter], best_parameter_list[lcv_parameter]
print "Evaluations:", genetic_parameters.counter

```

```
if __name__ == "__main__":  
    begin()
```

Appendix F. A program for using the parameters found by the program in Appendix IV or V to determine the number of trees and biomass in test images.

```
from Tkinter import *
import math
import random
import Image
import sys
import string
import genetic_parameters
import images
import gc
import array

def raw(image_number):
    in_image = Image.open(genetic_parameters.image_names[image_number])
    x_dim, y_dim = in_image.size
    images.x_dim.append(x_dim)
    images.y_dim.append(y_dim)
    images.raw_images.append(in_image)

def edge(image_number):
    edge_raw = Image.open(genetic_parameters.image_names[image_number])
    return edge_raw

def low_pass(source_image, image_number):
    lp_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    si_array = array.array("d")
    si_array.fromlist(list(source_image.getdata()))
    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]
    for lcv_row in range(0, y_dim):
        for lcv_column in range(0, x_dim):
            if lcv_row == 0 or lcv_row == images.y_dim[image_number] - 1 or lcv_column == 0 or lcv_column ==
images.x_dim[image_number] - 1:
```

```

        value = si_array[lcv_row * x_dim + lcv_column]
        lp_image.putpixel((lcv_column, lcv_row), value)
    else:
        value = si_array[lcv_row * x_dim + lcv_column]
        value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column - 1)]
        value = value + si_array[(lcv_row - 1) * x_dim + (lcv_column)]
        value = value + si_array[((lcv_row - 1) * x_dim + (lcv_column + 1))]
        value = value + si_array[((lcv_row) * x_dim + (lcv_column - 1))]
        value = value + si_array[((lcv_row) * x_dim + (lcv_column + 1))]
        value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column - 1))]
        value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column))]
        value = value + si_array[((lcv_row + 1) * x_dim + (lcv_column + 1))]
        value = float(value)/float(9)
        lp_image.putpixel((lcv_column, lcv_row), value)

return lp_image

def calculate_dog(SD):
    multipliers = []
    pi = math.pi
    sqrt2 = math.sqrt(2)
    for lcv_x in range(-5, 6):
        lcv_calculate = float(lcv_x)
        term_1 = -0.5 * sqrt2 / (math.sqrt(pi) * math.pow(SD, 3))
        term_2 = math.exp(-0.5 * ((lcv_calculate * lcv_calculate) / (SD * SD)))
        term_3 = 0.5 * math.sqrt(2) / (math.sqrt(pi) * math.pow(SD, 5))
        term_4 = (lcv_calculate) * (lcv_calculate)
        term_5 = math.exp(-0.5 * (lcv_calculate * lcv_calculate) / (SD * SD))
        coef = (term_1 * term_2) + (term_3 * term_4 * term_5)
        multipliers.append(coef)

return multipliers

def lmin(edge_raw, image_number, SD):

```

```

lm_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
lmax_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
x_dim = images.x_dim[image_number]
y_dim = images.y_dim[image_number]
multipliers = calculate_dog(SD)
edge_array = array.array("f")
edge_array.fromlist(list(edge_raw.getdata()))

pp_lm = lm_image.putpixel
pp_lmax = lmax_image.putpixel

for lcv_row in range (6, images.y_dim[image_number] - 6):
    for lcv_column in range (6, images.x_dim[image_number] - 6):
        value_01 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[lcv_column-1 + (lcv_row + 1) * x_dim] +
edge_array[lcv_column+1 + (lcv_row - 1) * x_dim]
        value_1 = edge_array[(lcv_column-5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column-6) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row - 6) * x_dim]
        value_2 = edge_array[(lcv_column-4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column-5) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row - 5) * x_dim]
        value_3 = edge_array[(lcv_column-3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column-4) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row - 4) * x_dim]
        value_4 = edge_array[(lcv_column-2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column-3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row - 3) * x_dim]
        value_5 = edge_array[(lcv_column-1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column-2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row - 2) * x_dim]
        value_6 = edge_array[(lcv_column+1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
        value_7 = edge_array[(lcv_column+2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
        value_8 = edge_array[(lcv_column+3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 2) * x_dim]
        value_9 = edge_array[(lcv_column+4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 3) * x_dim]
        value_10 = edge_array[(lcv_column+5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row + 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row + 4) * x_dim]
        value_11 = edge_array[(lcv_column) + (lcv_row-5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 5) * x_dim] +

```



```

edge_array[lcv_column + 1 + (lcv_row - 5) * x_dim]
    value_12 = edge_array[(lcv_column) + (lcv_row-4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 4) * x_dim]
    value_13 = edge_array[(lcv_column) + (lcv_row-3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 3) * x_dim]
    value_14 = edge_array[(lcv_column) + (lcv_row-2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 2) * x_dim]
    value_15 = edge_array[(lcv_column) + (lcv_row-1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row - 1) * x_dim]
    value_02 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row) * x_dim]
    value_16 = edge_array[(lcv_column) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
    value_17 = edge_array[(lcv_column) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 2) * x_dim]
    value_18 = edge_array[(lcv_column) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 3) * x_dim]
    value_19 = edge_array[(lcv_column) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 4) * x_dim]
    value_20 = edge_array[(lcv_column) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row + 5) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 5) * x_dim]
    value_21 = edge_array[(lcv_column-5) + (lcv_row) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 5 + (lcv_row + 1) * x_dim]
    value_22 = edge_array[(lcv_column-4) + (lcv_row) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 1) * x_dim]
    value_23 = edge_array[(lcv_column-3) + (lcv_row) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 1) * x_dim]
    value_24 = edge_array[(lcv_column-2) + (lcv_row) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 1) * x_dim]
    value_25 = edge_array[(lcv_column-1) + (lcv_row) * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 1) * x_dim]
    value_03 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + (lcv_row + 1) * x_dim]
    value_26 = edge_array[(lcv_column+1) + (lcv_row) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
    value_27 = edge_array[(lcv_column+2) + (lcv_row) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 1) * x_dim] +

```

```

edge_array[lcv_column + 2 + (lcv_row + 1) * x_dim]
    value_28 = edge_array[(lcv_column+3) + (lcv_row) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row + 1) * x_dim]
    value_29 = edge_array[(lcv_column+4) + (lcv_row) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row + 1) * x_dim]
    value_30 = edge_array[(lcv_column+5) + (lcv_row) * x_dim] + edge_array[(lcv_column + 5) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row + 1) * x_dim]
    value_31 = edge_array[(lcv_column-5) + (lcv_row+5) * x_dim] + edge_array[(lcv_column - 6) + (lcv_row + 4) * x_dim] +
edge_array[lcv_column - 4 + (lcv_row + 6) * x_dim]
    value_32 = edge_array[(lcv_column-4) + (lcv_row+4) * x_dim] + edge_array[(lcv_column - 5) + (lcv_row + 3) * x_dim] +
edge_array[lcv_column - 3 + (lcv_row + 5) * x_dim]
    value_33 = edge_array[(lcv_column-3) + (lcv_row+3) * x_dim] + edge_array[(lcv_column - 4) + (lcv_row + 2) * x_dim] +
edge_array[lcv_column - 2 + (lcv_row + 4) * x_dim]
    value_34 = edge_array[(lcv_column-2) + (lcv_row+2) * x_dim] + edge_array[(lcv_column - 3) + (lcv_row + 1) * x_dim] +
edge_array[lcv_column - 1 + (lcv_row + 3) * x_dim]
    value_35 = edge_array[(lcv_column-1) + (lcv_row+1) * x_dim] + edge_array[(lcv_column - 2) + (lcv_row) * x_dim] +
edge_array[lcv_column + (lcv_row + 2) * x_dim]
    value_04 = edge_array[lcv_column + lcv_row * x_dim] + edge_array[(lcv_column - 1) + (lcv_row - 1) * x_dim] +
edge_array[lcv_column + 1 + (lcv_row + 1) * x_dim]
    value_36 = edge_array[(lcv_column+1) + (lcv_row-1) * x_dim] + edge_array[(lcv_column) + (lcv_row - 2) * x_dim] +
edge_array[lcv_column + 2 + (lcv_row) * x_dim]
    value_37 = edge_array[(lcv_column+2) + (lcv_row-2) * x_dim] + edge_array[(lcv_column + 1) + (lcv_row - 3) * x_dim] +
edge_array[lcv_column + 3 + (lcv_row - 1) * x_dim]
    value_38 = edge_array[(lcv_column+3) + (lcv_row-3) * x_dim] + edge_array[(lcv_column + 2) + (lcv_row - 4) * x_dim] +
edge_array[lcv_column + 4 + (lcv_row - 2) * x_dim]
    value_39 = edge_array[(lcv_column+4) + (lcv_row-4) * x_dim] + edge_array[(lcv_column + 3) + (lcv_row - 5) * x_dim] +
edge_array[lcv_column + 5 + (lcv_row - 3) * x_dim]
    value_40 = edge_array[(lcv_column+5) + (lcv_row-5) * x_dim] + edge_array[(lcv_column + 4) + (lcv_row - 6) * x_dim] +
edge_array[lcv_column + 6 + (lcv_row - 4) * x_dim]

```

```
lm_list = []
```

```

lm_1 = multipliers[0]*value_1 + multipliers[1]*value_2 + multipliers[2]*value_3 + multipliers[3]*value_4 + multipliers[4] *
value_5 + multipliers[5] * value_01 + multipliers[6] * value_6 + multipliers[7]*value_7 + multipliers[8]*value_8 + multipliers[9] * value_9 + multipliers[10]
* value_10

```

```

lm_2 = multipliers[0]*value_11 + multipliers[1]*value_12 + multipliers[2]*value_13 + multipliers[3]*value_14 + multipliers[4] *

```

```

value_15 + multipliers[5] * value_02 + multipliers[6] * value_16 + multipliers[7]*value_17 + multipliers[8]*value_18 + multipliers[9] * value_19 +
multipliers[10] * value_20
    lm_3 = multipliers[0]*value_21 + multipliers[1]*value_22 + multipliers[2]*value_23 + multipliers[3]*value_24 + multipliers[4] *
value_25 + multipliers[5] * value_03 + multipliers[6] * value_26 + multipliers[7]*value_27 + multipliers[8]*value_28 + multipliers[9] * value_29 +
multipliers[10] * value_30
    lm_4 = multipliers[0]*value_31 + multipliers[1]*value_32 + multipliers[2]*value_33 + multipliers[3]*value_34 + multipliers[4] *
value_35 + multipliers[5] * value_04 + multipliers[6] * value_36 + multipliers[7]*value_37 + multipliers[8]*value_38 + multipliers[9] * value_39 +
multipliers[10] * value_40
    lm_list.append(lm_1)
    lm_list.append(lm_2)
    lm_list.append(lm_3)
    lm_list.append(lm_4)
    lm_list.sort()
    pp_lm((lcv_column, lcv_row), lm_list[3])
    pp_lmax((lcv_column, lcv_row), lm_list[0])
return lm_image, lmax_image

```

```

def make_int_maps(image_number):
    images.value_image = Image.new("F", (images.x_dim[image_number], images.y_dim[image_number]), 0)
    images.radius_image = Image.new("L", (images.x_dim[image_number], images.y_dim[image_number]), 0)

```

```

def find_value_diam(image_number, center_thresh, ratio_thresh, ae_thresh, min_total, decay):

```

```

    edge_array = array.array("f")
    edge_array.fromlist(list(images.edge_images[image_number].getdata()))

```

```

    lmin_array = array.array("f")
    lmin_array.fromlist(list(images.lmin_image.getdata()))

```

```

    raw_array = array.array("f")
    raw_array.fromlist(list(images.raw_images[image_number].getdata()))

```

```

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

```

```

    pp_value = images.value_image.putpixel

```

```

pp_radius = images.radius_image.putpixel

for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        test_1 = lmin_array[lcv_line * x_dim + lcv_column]
        test_2 = raw_array[lcv_line * x_dim + lcv_column]
        if test_1 < center_thresh and test_2 > ratio_thresh:
            high_value = -999
            high_radius = 0
            stop = "N"
            stop_now = "N"
            for lcv_radius in range(genetic_parameters.min_radius, genetic_parameters.max_radius + 1):
                if stop == "Y":
                    stop_now = "Y"
                pixels = 0
                value = 0
                sqrad = lcv_radius * lcv_radius
                for lcv_y in range (-1 * lcv_radius, lcv_radius + 1):
                    x_radius = int(round(math.sqrt(sqrad - lcv_y * lcv_y)))
                    position_y = lcv_line + lcv_y
                    pos_position_x = lcv_column + x_radius
                    neg_position_x = lcv_column - x_radius
                    if (test_2 - edge_array[position_y * x_dim + pos_position_x]) > ae_thresh:
                        stop = "Y"
                    if (test_2 - edge_array[position_y * x_dim + neg_position_x]) > ae_thresh:
                        stop = "Y"
                    value = value + lmin_array[position_y * x_dim + pos_position_x]
                    value = value + lmin_array[position_y * x_dim + neg_position_x]
                    if x_radius == 0:
                        pixels = pixels + 1
                    else:
                        pixels = pixels + 2

            total_value = float(value) / float(pixels)
            total_value = total_value + decay*lcv_radius
            if total_value > high_value:

```

```

        high_value = total_value
        high_radius = lcv_radius
    if stop_now == "Y":
        break
    if high_value < min_total:
        high_value = 0

    pp_value((lcv_column, lcv_line), high_value)
    pp_radius((lcv_column, lcv_line), high_radius)

```

```
def refine_trees(image_number, alpha, beta, gamma):
```

```

    x_dim = images.x_dim[image_number]
    y_dim = images.y_dim[image_number]

    lmax_array = array.array("f")
    lmax_array.fromlist(list(images.lmax_image.getdata()))

    radius_array = array.array("f")
    radius_array.fromlist(list(images.radius_image.getdata()))

    value_array = array.array("f")
    value_array.fromlist(list(images.value_image.getdata()))

    raw_array = array.array("f")
    raw_array.fromlist(list(images.raw_images[image_number].getdata()))

    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
            if pixel_rad > 0:
                keep = 1
                score = alpha * value_array[lcv_line * x_dim + lcv_column]
                score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
                score = score + gamma * raw_array[lcv_line * x_dim + lcv_column]

```

```

for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
    for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
        radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
        ignore = "N"
        if round(radius) > pixel_rad:
            ignore = "Y"
        if ignore == "N":
            wscore = alpha * value_array[win_y * x_dim + win_x]
            wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
            wscore = wscore + gamma * raw_array[win_y * x_dim + win_x]
            if wscore > score:
                keep = 0
                break
        if int(keep) == 0:
            break
    images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)

```

```

radius_array = array.array("f")
radius_array.fromlist(list(images.radius_image.getdata()))

```

#The following routine cleans up any small trees that are encroaching on a large tree's space

```

for lcv_line in range(genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
    for lcv_column in range(genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
        pixel_rad = radius_array[lcv_line * x_dim + lcv_column]
        if pixel_rad > 0:
            keep = 1
            score = alpha * value_array[lcv_line * x_dim + lcv_column]
            score = score + beta * lmax_array[lcv_line * x_dim + lcv_column]
            score = score + gamma * raw_array[lcv_line * x_dim + lcv_column]
            for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                    radius = math.sqrt((win_y - lcv_line) * (win_y - lcv_line) + (win_x - lcv_column) * (win_x - lcv_column))
                    ignore = "N"
                    if round(radius) > pixel_rad:
                        ignore = "Y"

```

```

if ignore == "N" and radius_array[win_y * x_dim + win_x] > 0:
    wscore = alpha * value_array[win_y * x_dim + win_x]
    wscore = wscore + beta * lmax_array[win_y * x_dim + win_x]
    wscore = wscore + gamma * raw_array[win_y * x_dim + win_x]
    if wscore < score:
        images.radius_image.putpixel((win_x, win_y), 0)

```

```

def refine_trees_old(image_number, alpha, beta, gamma, examination_fraction):
    if examination_fraction > 1:
        examination_fraction = float(1)
    for lcv_line in range (genetic_parameters.max_radius, images.y_dim[image_number] - genetic_parameters.max_radius):
        for lcv_column in range (genetic_parameters.max_radius, images.x_dim[image_number] - genetic_parameters.max_radius):
            pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))
            if pixel_rad > 0:
                keep = 1
                score = alpha * images.value_image.getpixel((lcv_column, lcv_line))
                score = score + beta * images.lmin_image.getpixel((lcv_column, lcv_line))
                score = score + gamma * images.raw_images[image_number].getpixel((lcv_column, lcv_line))
                for win_x in range (lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
                    for win_y in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
                        radius = math.sqrt((win_x - lcv_line) * (win_x - lcv_line) + (win_y - lcv_column) * (win_y - lcv_column))
                        ignore = "N"
                        if round(radius) > pixel_rad * examination_fraction:
                            ignore = "Y"
                        if ignore == "N":
                            wscore = alpha * images.value_image.getpixel((win_y, win_x))
                            wscore = wscore + beta * images.lmin_image.getpixel((win_y, win_x))
                            wscore = wscore + gamma * images.raw_images[image_number].getpixel((win_y, win_x))
                            if wscore > score:
                                keep = 0
                                break
            if int(keep) == 0:
                break

```

```
images.radius_image.putpixel((lcv_column, lcv_line), pixel_rad * keep)
```

```
def find_height(lcv_column, lcv_line, pixel_rad, image_number):
    max_height = -2000
    for win_y in range(lcv_line - pixel_rad, lcv_line + pixel_rad + 1):
        for win_x in range(lcv_column - pixel_rad, lcv_column + pixel_rad + 1):
            radius = math.sqrt((win_x - lcv_column) * (win_x - lcv_column) + (win_y - lcv_line) * (win_y - lcv_line))
            if round(radius) <= pixel_rad:
                lidar_value = images.raw_images[image_number].getpixel((win_x, win_y))
                if lidar_value > max_height:
                    max_height = lidar_value

    if max_height == -2000:
        print pixel_rad
    return max_height

def make_output(image_number):
    trees = 0
    diameter_list = []
    height_list = []
    for lcv_line in range(genetic_parameters.buffer, images.y_dim[image_number] - genetic_parameters.buffer):
        for lcv_column in range(genetic_parameters.buffer, images.x_dim[image_number] - genetic_parameters.buffer):
            pixel_rad = images.radius_image.getpixel((lcv_column, lcv_line))
            center_x = float(images.x_dim[image_number]) / float(2)
            center_y = float(images.y_dim[image_number]) / float(2)
            term_1 = (center_x - float(lcv_column)) * (center_x - float(lcv_column))
            term_2 = (center_y - float(lcv_line)) * (center_y - float(lcv_line))
            distance = math.sqrt(term_1 + term_2)
            if pixel_rad > 0 and int(round(distance)) <= genetic_parameters.plot_radius:
                trees = trees + 1
                diameter = genetic_parameters.pixel_length + 2 * genetic_parameters.pixel_length * pixel_rad
                diameter_list.append(diameter)
                height = find_height(lcv_column, lcv_line, pixel_rad, image_number)
                height_list.append(height)

    area = (math.pi * math.pow(genetic_parameters.plot_radius * genetic_parameters.pixel_length, 2)) / 10000
    density = trees / area
```



```

genetic_parameters.tree_count.append(trees)
genetic_parameters.tree_density.append(density)
genetic_parameters.crown_widths.append(diameter_list)
genetic_parameters.tree_heights.append(height_list)

```

```

def start(image_number, alpha, beta, gamma, center_thresh, raw_thresh, ae_thresh, min_total, decay, SD):

```

```

    genetic_parameters.counter = genetic_parameters.counter + 1
    if genetic_parameters.first_run[image_number] == "Y":
        raw(image_number)
    if genetic_parameters.first_run[image_number] == "Y":
        images.lp_images.append(low_pass(images.raw_images[image_number], image_number))
    if genetic_parameters.first_run[image_number] == "Y":
        if genetic_parameters.edge_method == "R":
            images.edge_images.append(lp_image)
        else:
            edge_raw = edge(image_number)
            images.edge_images.append(low_pass(edge_raw, image_number))
    else:
        edge_raw = edge(image_number)
    images.lmin_image, images.lmax_image = lmin(edge_raw, image_number, SD)
    make_int_maps(image_number)
    find_value_diam(image_number, center_thresh, ratio_thresh, ae_thresh, min_total, decay)
    refine_trees(image_number, alpha, beta, gamma)
    make_output(image_number)
    del images.radius_image
    del images.value_image

```

```

def eval_func (parameter_list):

```

```

    genetic_parameters.crown_widths = []
    genetic_parameters.tree_density = []
    genetic_parameters.tree_count = []
    genetic_parameters.tree_heights = []

    genetic_parameters.first_run = []

```

```

for lcv in range (0, genetic_parameters.number_test_images):
    genetic_parameters.first_run.append("Y")

for lcv_image in range (0, genetic_parameters.number_test_images):
    alpha = parameter_list[0]
    beta = parameter_list[1]
    gamma = parameter_list[2]
    center = parameter_list[3]
    ratio = parameter_list[4]
    ae = parameter_list[5]
    min = parameter_list[6]
    decay = parameter_list[7]
    SD = parameter_list[8]
    start(lcv_image, alpha, beta, gamma, center, ratio, ae, min, decay, SD)
genetic_parameters.first_run[lcv_image] = "N"

p1 = genetic_parameters.initial_equation_parameters[0]
p2 = genetic_parameters.initial_equation_parameters[1]
p3 = genetic_parameters.initial_equation_parameters[2]

print "======"

ofile = open("results.csv", "w")

for lcv_stand in range (0, genetic_parameters.number_test_images):
    biomass_sum = 0
    height_list = []
    for lcv_tree in range(0, genetic_parameters.tree_count[lcv_stand]):
        term_1 = p1 * math.log(genetic_parameters.tree_density[lcv_stand])
        height_list.append(genetic_parameters.tree_heights[lcv_stand][lcv_tree])
        if genetic_parameters.tree_heights[lcv_stand][lcv_tree] > 0:
            term_2 = p2 * math.log(genetic_parameters.tree_heights[lcv_stand][lcv_tree])
        else:
            term_2 = 0

```

```

        tree_biomass = math.exp(term_1 + term_2 + p3)
        biomass_sum = biomass_sum + tree_biomass
    trees = genetic_parameters.tree_count[lcv_stand]
    if trees < 1:
        trees = 1
    height_list.sort()
    number_25 = int(0.25 * genetic_parameters.tree_count[lcv_stand])
    number_50 = int(0.50 * genetic_parameters.tree_count[lcv_stand])
    number_75 = int(0.75 * genetic_parameters.tree_count[lcv_stand])
    #print "Trees in stand =", genetic_parameters.tree_count[lcv_stand]
    try:
        output_line = str(lcv_stand) + "," + str(genetic_parameters.tree_count[lcv_stand]) + "," + str(biomass_sum) + "," +
str(height_list[number_25]) + "," + str(height_list[number_50]) + "," + str(height_list[number_75]) + "\n"
    except:
        output_line = str(lcv_stand) + "," + str(genetic_parameters.tree_count[lcv_stand]) + "," + str(biomass_sum) + "," + str(0) + "," +
str(0) + "," + str(0) + "\n"
        ofile.write(output_line)
    ofile.close()

def begin():
#
#This function will start the program out if it is being run as a stand alone program
#

    genetic_parameters.print_biomass_estimates = "Y"

    genetic_parameters.output_file = "c:/temp/pca_3_results.txt"

    parameter_names = ["Alpha:", "Beta:", "Gamma:", "Center thresh.:", "Ratio thresh.:", "A. E. thresh.:", "Min. total:", "Decay:", "SD"]

    genetic_parameters.number_test_images = 25

    genetic_parameters.image_names = []

```

```
genetic_parameters.pixel_length = 0.5
```

```
genetic_parameters.plot_radius = 30
```

```
genetic_parameters.image_names.append("1_test.tif")  
genetic_parameters.image_names.append("2_test.tif")  
genetic_parameters.image_names.append("3_test.tif")  
genetic_parameters.image_names.append("4_test.tif")  
genetic_parameters.image_names.append("5_test.tif")  
genetic_parameters.image_names.append("6_test.tif")  
genetic_parameters.image_names.append("7_test.tif")  
genetic_parameters.image_names.append("8_test.tif")  
genetic_parameters.image_names.append("9_test.tif")  
genetic_parameters.image_names.append("10_test.tif")  
genetic_parameters.image_names.append("11_test.tif")  
genetic_parameters.image_names.append("12_test.tif")  
genetic_parameters.image_names.append("13_test.tif")  
genetic_parameters.image_names.append("14_test.tif")  
genetic_parameters.image_names.append("15_test.tif")  
genetic_parameters.image_names.append("16_test.tif")  
genetic_parameters.image_names.append("17_test.tif")  
genetic_parameters.image_names.append("18_test.tif")  
genetic_parameters.image_names.append("19_test.tif")  
genetic_parameters.image_names.append("20_test.tif")  
genetic_parameters.image_names.append("21_test.tif")  
genetic_parameters.image_names.append("22_test.tif")  
genetic_parameters.image_names.append("23_test.tif")  
genetic_parameters.image_names.append("24_test.tif")  
genetic_parameters.image_names.append("25_test.tif")
```

```
genetic_parameters.verbose = "N"
```

```
ofile = open(genetic_parameters.output_file, "w")
```

```
output_string = "Iteration,Best score,Prob. crossover,Prob. avg_crossover,Prob. mutation\n"
```

```
ofile.write(output_string)
```

```
ofile.write(output_string)
```

```
images.raw_images = []
```

```
images.lmin_images = []
```

```
images.lmax_images = []
```

```
images.x_dim = []
```

```
images.y_dim = []
```

```
images.lp_images = []
```

```
images.edge_images = []
```

```
genetic_parameters.buffer = 10
```

```
genetic_parameters.edge_method = "I"
```

```
genetic_parameters.filter = "L"
```

```
genetic_parameters.min_radius = 1
```

```
genetic_parameters.max_radius = 8
```

```
genetic_parameters.number_of_parameters = 9
```

```
genetic_parameters.counter = 0
```

#The following list of parameters is obtained from the program in Appendix IV. For the program in Appendix V use dummy values and ignore the biomass predictions.

```
genetic_parameters.initial_equation_parameters = [-2.035005758001514, 6.5709069357706085, 3.3244155751847932]
```

#The following list of parameters is obtained from the programs in Appendix IV and V

```
parameter_list = [1.1255602331188657, -7.9257101142078863, 2.3789367501738203, 3.0544628766428001, 6.6803593635939462,  
2.7333777127799705, 6.3879022189706491, -13.986968870249489, 1.7841668251377372]
```

```
eval_func(parameter_list)
```

```
if __name__ == "__main__":  
    begin()
```

Appendix G. Measurements for the trees in each plot whose DBH and dominance status were measured. DBH is given in centimeters. D indicates that the tree is dominant, and S indicates that the tree is subdominant. The predicted biomass is given in grams.

Plot 1

Diameter	Position	Biomass		Diameter	Position	Biomass
7.6	S	11583		14.0	D	45088
7.8	S	12280		14.2	D	46692
8.0	S	13000		14.2	D	46692
8.3	S	14123		14.9	D	52569
8.4	S	14509		14.9	D	52569
8.5	S	14900		15.5	D	57940
8.8	S	16110		15.6	D	58865
8.8	S	16110		15.9	D	61694
8.9	S	16524		16.0	D	62654
8.9	S	16524		16.0	D	62654
9.8	S	20524		16.2	D	64602
10.4	S	23460		16.5	D	67590
10.5	S	23970		16.5	D	67590
11.1	S	27163		16.8	D	70658
11.2	D	26018		16.8	D	70658
11.3	D	26594		17.4	D	77039
11.3	S	28277		17.4	D	77039
11.9	D	30210		17.5	D	78135
12.2	D	32121		17.9	D	82609
12.3	D	32774		18.3	D	87233
12.6	D	34779		18.6	D	90799
12.6	D	34779		18.6	D	90799
12.8	D	36155		20.9	D	121015
13.1	D	38279		22.9	D	151577
13.5	D	41223				

Plot 2:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.3	S	10580		16.5	D	67590
7.4	S	10909		16.5	D	67590
7.7	S	11929		16.5	D	67590
8.3	S	14123		16.7	D	69626
8.5	S	14900		16.7	D	69626
9.6	D	17796		16.9	D	71699
9.7	S	20056		17.0	D	72749
10.0	S	21478		17.1	D	73808
10.4	S	23460		17.4	D	77039
10.5	S	23970		17.6	D	79240
10.9	S	26074		17.7	D	80354
11.0	S	26615		17.9	D	82609
12.0	S	32371		17.9	D	82609
12.9	D	36855		18.0	D	83751
13.8	D	43517		18.2	D	86063
14.2	D	46692		18.5	D	89601
14.4	D	48329		18.6	D	90799
14.6	D	49999		18.7	D	92006
14.7	D	50848		18.8	D	93223
15.1	D	54325		18.9	D	94450
15.4	D	57023		19.1	D	96932
15.4	D	57023		19.2	D	98187
15.5	D	57940		20.9	D	121015
15.8	D	60742		21.3	D	126802
15.9	D	61694		21.8	D	134263
16.0	D	62654		22.0	D	137319
16.1	D	63624		22.4	D	143553
16.3	D	65589		22.5	D	145137

Plot 3:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.3	S	10580		15.8	D	60742
7.4	S	10909		15.9	D	61694
7.7	S	11929		16.0	D	62654
8.3	S	14123		16.1	D	63624
8.5	S	14900		16.3	D	65589
9.6	D	17796		18.9	D	94450
9.7	S	20056		19.3	D	99452
10.0	S	21478		19.3	D	99452
10.4	S	23460		19.4	D	100726
10.5	S	23970		19.9	D	107244
10.9	S	26074		19.9	D	107244
11.0	S	26615		19.9	D	107244
12.0	S	32371		20.1	D	109920
12.9	D	36855		20.5	D	115388
13.8	D	43517		20.6	D	116780
14.2	D	46692		20.7	D	118182
14.4	D	48329		21.0	D	122447
14.6	D	49999		21.4	D	128274
14.7	D	50848		21.5	D	129756
15.1	D	54325		21.9	D	135786
15.4	D	57023		22.5	D	145137
15.4	D	57023		23.5	D	161552
15.5	D	57940				

Plot 4:

Diameter	Position	Biomass		Diameter	Position	Biomass
8.1	S	13369		15.6	D	58865
8.2	S	13743		15.6	D	58865
8.3	S	14123		15.8	D	60742
8.6	S	15298		15.9	D	61694
9.6	S	19593		16.0	D	62654
9.7	D	18256		16.0	S	61840
10.4	S	23460		16.2	D	64602
10.8	S	25539		16.3	D	65589
11.1	S	27163		16.4	D	66585
11.3	D	26594		17.0	D	72749
11.4	D	27178		17.0	D	72749
11.8	S	31170		17.2	D	74876
12.0	D	30839		17.2	D	74876
12.0	D	30839		17.3	D	75953
12.1	D	31476		17.4	D	77039
12.4	D	33434		17.4	D	77039
12.8	D	36155		17.4	D	77039
13.4	D	40475		17.6	D	79240
13.6	D	41980		17.6	D	79240
13.8	D	43517		17.8	D	81477
14.4	D	48329		18.0	D	83751
14.5	D	49160		18.2	D	86063
14.6	D	49999		18.5	D	89601
14.8	D	51704		18.8	D	93223
14.9	D	52569		19.0	D	95686
15.0	D	53443		19.5	D	102011
15.1	D	54325		19.9	D	107244
15.2	D	55216		20.2	D	111272
15.5	D	57940		20.4	D	114006

Plot 5

Diameter	Position	Biomass		Diameter	Position	Biomass
7.9	S	12637		15.8	D	60742
8.6	S	15298		15.9	D	61694
9.8	D	18723		16.0	D	62654
9.9	S	20998		16.2	D	64602
10.2	S	22457		16.3	D	65589
10.4	S	23460		16.7	D	69626
11.2	D	26018		16.8	D	70658
11.2	S	27717		16.9	D	71699
11.9	D	30210		16.9	D	71699
11.9	D	30210		17.0	D	72749
12.0	S	32371		17.1	D	73808
12.4	S	34850		17.2	D	74876
12.5	D	34103		17.2	D	74876
13.8	D	43517		17.2	D	74876
13.8	D	43517		17.3	D	75953
14.1	D	45886		17.4	D	77039
14.2	D	46692		17.4	D	77039
14.4	D	48329		17.8	D	81477
14.5	D	49160		17.8	D	81477
14.6	D	49999		17.8	D	81477
14.7	D	50848		17.8	D	81477
14.9	D	52569		18.2	D	86063
15.1	D	54325		18.5	D	89601
15.3	D	56115		18.8	D	93223
15.3	D	56115		18.8	D	93223
15.4	D	57023		18.9	D	94450
15.6	D	58865		21.2	D	125341
15.7	D	59799		21.6	D	131248

Plot 6

Diameter	Position	Biomass		Diameter	Position	Biomass
7.2	S	10256		13.0	D	37563
7.5	S	11243		13.4	D	40475
7.9	S	12637		13.6	D	41980
8.0	S	13000		13.7	D	42744
8.1	S	13369		14.0	D	45088
8.1	S	13369		14.0	D	45088
8.2	S	13743		14.5	D	49160
8.3	S	14123		14.5	D	49160
8.8	S	16110		14.8	D	51704
8.9	S	16524		15.2	D	55216
8.9	S	16524		15.2	D	55216
9.1	S	17372		15.9	D	61694
9.2	S	17804		16.3	D	65589
9.2	S	17804		16.4	D	66585
9.2	S	17804		16.7	D	69626
9.3	S	18243		16.7	D	69626
9.4	S	18687		16.8	D	70658
9.9	S	20998		16.8	D	70658
10.1	S	21965		16.9	D	71699
10.2	D	20663		17.3	D	75953
10.2	S	22457		17.5	D	78135
10.3	S	22955		17.7	D	80354
10.4	S	23460		17.8	D	81477
10.6	S	24487		17.9	D	82609
10.7	S	25010		17.9	D	82609
10.9	S	26074		18.2	D	86063
11.0	S	26615		18.4	D	88412
11.0	S	26615		18.9	D	94450
11.1	D	25449		19.1	D	96932
11.1	D	25449		19.3	D	99452
11.2	S	27717		19.5	D	102011
11.5	S	29415		19.9	D	107244
12.0	S	32371		20.3	D	112634
12.5	D	34103		20.9	D	121015
12.9	D	36855		21.2	D	125341
12.9	D	36855				

Plot 7

Diameter	Position	Biomass		Diameter	Position	Biomass
9.5	D	17342		16.1	D	63624
9.7	D	18256		16.1	D	63624
10.1	D	20167		16.4	D	66585
12.3	D	32774		16.5	D	67590
12.5	D	34103		16.5	D	67590
12.5	S	35485		16.5	D	67590
12.6	D	34779		16.6	D	68604
12.6	D	34779		16.6	D	68604
13.5	D	41223		16.7	D	69626
13.8	D	43517		16.8	D	70658
14.2	D	46692		17.0	D	72749
14.3	D	47506		17.1	D	73808
14.4	D	48329		17.2	D	74876
14.9	D	52569		17.2	D	74876
15.0	D	53443		17.3	D	75953
15.0	D	53443		17.5	D	78135
15.1	D	54325		17.5	D	78135
15.2	D	55216		17.9	D	82609
15.2	D	55216		17.9	D	82609
15.3	D	56115		18.0	D	83751
15.5	D	57940		18.2	D	86063
15.7	D	59799		19.6	D	103304
16.0	D	62654		19.9	D	107244

Plot 8

Diameter	Position	Biomass
9.5	D	17342
9.7	S	20056
10.5	D	22193
10.7	D	23249
11.0	D	24888
12.4	D	33434
13.0	D	37563
13.1	D	38279
13.3	D	39735
13.5	D	41223
13.7	D	42744
13.7	D	42744
14.2	D	46692
14.4	D	48329
14.5	D	49160
14.8	D	51704
15.0	D	53443
15.1	D	54325
15.9	D	61694
16.0	D	62654
16.1	D	63624
16.1	D	63624
16.2	D	64602
16.2	D	64602
16.4	D	66585
16.5	D	67590
16.5	D	67590

Plot 9:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.0	S	9627		14.1	D	45886
7.2	S	10256		14.2	D	46692
7.4	S	10909		14.2	D	46692
7.4	S	10909		14.3	D	47506
7.5	S	11243		14.3	D	47506
8.1	S	13369		14.4	D	48329
8.8	S	16110		14.5	D	49160
8.8	S	16110		14.5	D	49160
8.9	S	16524		15.0	D	53443
9.2	S	17804		15.0	D	53443
9.2	S	17804		15.3	D	56115
9.2	S	17804		15.4	D	57023
9.2	S	17804		15.5	D	57940
9.6	S	19593		15.5	D	57940
9.8	S	20524		15.7	D	59799
9.8	S	20524		16.1	D	63624
9.8	S	20524		16.4	D	66585
10.8	D	23788		16.5	D	67590
10.8	D	23788		16.5	D	67590
11.3	D	26594		16.6	D	68604
11.4	S	28843		16.6	D	68604
11.7	S	30579		16.6	D	68604
11.8	D	29588		16.8	D	70658
12.0	D	30839		17.0	D	72749
12.1	D	31476		17.1	D	73808
12.2	D	32121		17.4	D	77039
12.3	D	32774		17.5	D	78135
12.3	D	32774		17.8	D	81477
12.4	D	33434		17.9	D	82609
13.0	D	37563		18.3	D	87233
13.3	D	39735		18.9	D	94450
13.4	D	40475		19.0	D	95686
13.8	D	43517		19.8	D	105921
13.9	D	44298				

Plot 10:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.6	S	11583		14.8	D	51704
10.3	D	21166		14.8	D	51704
10.6	S	24487		14.8	D	51704
10.6	S	24487		15.2	D	55216
11.0	S	26615		15.3	D	56115
11.2	S	27717		15.4	D	57023
11.5	S	29415		15.6	D	58865
11.6	S	29994		15.6	D	58865
11.8	S	31170		15.7	D	59799
11.9	D	30210		15.7	D	59799
12.6	D	34779		15.9	D	61694
13.0	D	37563		16.1	D	63624
13.2	D	39003		16.3	D	65589
13.3	D	39735		16.5	D	67590
13.4	D	40475		16.8	D	70658
13.6	D	41980		16.9	D	71699
13.7	D	42744		17.0	D	72749
13.8	D	43517		17.1	D	73808
13.9	D	44298		17.3	D	75953
14.1	D	45886		17.3	D	75953
14.2	D	46692		17.5	D	78135
14.2	D	46692		17.7	D	80354
14.2	D	46692		17.9	D	82609
14.2	D	46692		17.9	D	82609
14.6	D	49999		18.3	D	87233
14.6	D	49999				

Plot 11:

Diameter	Position	Biomass
9.3	S	18243
9.6	S	19593
11.5	D	27769
11.7	D	28974
11.9	D	30210
12.0	D	30839
12.4	D	33434
12.6	D	34779
12.6	D	34779
12.6	D	34779
12.9	D	36855
13.3	D	39735
13.5	D	41223
13.8	D	43517
13.9	D	44298
13.9	D	44298
14.1	D	45886
14.1	D	45886
14.3	D	47506
14.6	D	49999
14.7	D	50848
14.8	D	51704
15.4	D	57023
15.8	D	60742
15.8	D	60742
15.8	D	60742
15.9	D	61694
16.1	D	63624
16.2	D	64602
17.2	D	74876
17.5	D	78135
18.3	D	87233
18.5	D	89601

Plot 13:

Diameter	Position	Biomass		Diameter	Position	Biomass
9	S	16945		15	D	53443
9	S	16945		15	D	53443
11	D	24888		15	D	53443
11	S	26615		15	D	53443
12	D	30839		15	D	53443
12	D	30839		15	D	53443
12	S	32371		15	D	53443
13	D	37563		16	D	62654
13	D	37563		16	D	62654
13	D	37563		16	D	62654
13	D	37563		16	D	62654
13	D	37563		16	D	62654
13	D	37563		16	D	62654
13	D	37563		16	D	62654
13	D	37563		16	D	62654
13	D	37563		17	D	72749
13	D	37563		17	D	72749
13	S	38759		17	D	72749
13	S	38759		17	D	72749
14	D	45088		17	D	72749
14	D	45088		17	D	72749
14	D	45088		17	D	72749
14	D	45088		18	D	83751
14	D	45088		18	D	83751
14	D	45088		18	D	83751
14	D	45088		18	D	83751
14	D	45088		19	D	95686
14	D	45088		19	D	95686
15	D	53443		19	D	95686
15	D	53443		19	D	95686
15	D	53443		19	D	95686

Plot 14:

Diameter	Position	Biomass		Diameter	Position	Biomass
8.2	S	13743		15.8	D	60742
8.3	S	14123		15.9	D	61694
8.3	S	14123		16.1	D	63624
9.5	S	19137		16.1	D	63624
10.0	S	21478		16.4	D	66585
10.2	S	22457		16.4	D	66585
10.5	S	23970		16.5	D	67590
10.9	S	26074		16.7	D	69626
13.0	D	37563		16.7	D	69626
13.1	D	38279		16.7	D	69626
13.2	D	39003		17.0	D	72749
13.7	D	42744		17.1	D	73808
13.9	D	44298		17.2	D	74876
14.3	D	47506		17.5	D	78135
14.5	D	49160		17.5	D	78135
14.6	D	49999		17.6	D	79240
15.0	D	53443		17.9	D	82609
15.1	D	54325		17.9	D	82609
15.1	D	54325		18.3	D	87233
15.2	D	55216		18.3	D	87233
15.3	D	56115		18.6	D	90799
15.3	D	56115		18.7	D	92006
15.3	D	56115		18.7	D	92006
15.4	D	57023		18.8	D	93223
15.5	D	57940		19.3	D	99452
15.5	D	57940		19.4	D	100726
15.6	D	58865		19.5	D	102011
15.6	D	58865		19.5	D	102011
15.6	D	58865		19.6	D	103304
15.7	D	59799		21.7	D	132751

Plot 15:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.0	S	8172		14.1	D	45886
7.6	S	10007		14.5	D	49554
7.7	S	10335		14.6	D	50326
7.8	S	10669		15.1	D	54287
8.1	S	11709		15.3	D	56115
8.1	S	11709		15.9	D	61694
8.3	S	12434		15.9	D	60974
8.5	S	13185		16.1	D	63624
8.9	S	14767		16.2	D	64602
8.9	S	14767		16.4	D	66585
8.9	S	14767		16.5	D	67590
9.0	S	15179		16.5	D	66273
9.1	S	15598		16.9	D	71699
9.3	S	16457		17.2	D	72768
9.4	S	16896		17.4	D	74685
9.9	S	19198		17.5	D	75655
9.9	S	19198		17.6	D	76631
10.0	S	21478		17.7	D	80354
10.2	S	22457		18.0	D	83751
10.3	S	22955		18.5	D	85731
10.4	S	23460		18.8	D	93223
11.2	S	27717		18.8	D	88890
11.3	S	28277		19.0	D	91032
11.5	S	29415		19.1	D	96932
11.7	D	28974		19.1	D	92114
12.0	D	32371		19.2	D	98187
13.0	D	37563		19.3	D	94298
13.1	D	38279		21.1	D	123889
13.4	D	40475		21.7	D	122754
13.9	D	44298		23.8	D	151112

Plot 16:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.5	S	11243		16.4	D	66585
7.6	S	11583		16.5	D	67590
7.8	S	12280		16.6	D	68604
8.3	S	14123		16.8	D	70658
8.3	S	14123		17.0	D	72749
8.3	S	14123		17.2	D	74876
8.4	S	14509		17.2	D	74876
8.5	S	14900		17.2	D	74876
8.8	S	16110		17.5	D	78135
9.0	S	16945		17.8	D	81477
9.1	S	17372		17.9	D	82609
9.2	S	17804		18.4	D	88412
9.6	S	19593		18.5	D	89601
9.7	S	20056		18.6	D	90799
11.0	D	24888		19.0	D	95686
11.4	D	27178		19.2	D	98187
12.5	D	34103		19.3	D	99452
12.6	D	34779		19.4	D	100726
13.4	D	40475		19.6	D	103304
13.6	D	41980		20.4	D	114006
15.1	D	54325		20.5	D	115388
15.3	D	56115		21.0	D	122447
15.5	D	57940		21.1	D	123889
15.7	D	59799		21.3	D	126802
15.7	D	59799		21.4	D	128274
15.7	D	59799		21.6	D	131248
15.9	D	61694		21.8	D	134263
16.3	D	65589		22.9	D	151577
16.4	D	66585		23.3	D	158185
16.4	D	66585		24.4	D	177226

Plot 17:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.6	S	11583		16.4	D	66585
7.8	S	12280		16.6	D	68604
8.7	S	15701		16.7	D	69626
8.9	S	16524		16.8	D	70658
9.8	S	20524		16.9	D	71699
9.8	S	20524		17.1	D	73808
10.0	S	21478		17.2	D	74876
10.0	S	21478		17.6	D	79240
11.1	D	25449		18.0	D	83751
12.3	D	32774		18.0	D	83751
12.3	D	34221		18.1	D	84902
12.5	D	34103		18.1	D	84902
12.7	D	35463		18.3	D	87233
13.2	D	39003		18.9	D	94450
13.2	D	39003		19.0	D	95686
13.5	D	41223		19.1	D	96932
13.7	D	42744		19.2	D	98187
13.9	D	44298		19.2	D	98187
14.2	D	46692		19.3	D	99452
14.2	D	46692		19.4	D	100726
14.5	D	49160		19.5	D	102011
14.7	D	50848		19.6	D	103304
14.8	D	51704		19.8	D	105921
15.1	D	54325		20.1	D	109920
15.3	D	56115		20.3	D	112634
15.5	D	57940		20.3	D	112634
15.6	D	58865		20.5	D	115388
16.2	D	64602		21.5	D	129756
16.3	D	65589		21.8	D	134263
16.4	D	66585		23.2	D	156517

Plot 18:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.4	D	9371		13.5	D	41223
7.4	D	9371		13.5	D	41223
7.5	D	9686		13.5	D	41223
7.5	S	11243		14.0	D	45088
7.8	S	12280		14.2	D	46692
7.9	D	11009		14.2	D	46692
8.0	D	11356		14.8	D	51704
8.1	D	11709		14.8	D	51704
8.1	S	13369		14.8	D	51704
8.3	D	12434		15.3	D	56115
8.4	D	12806		15.8	D	60742
8.4	S	14509		16.0	D	62654
8.6	S	15298		16.2	D	64602
9.0	D	15179		16.3	D	65589
9.5	D	17342		16.4	D	66585
9.6	D	17796		16.8	D	70658
10.2	D	20663		16.9	D	71699
10.2	D	20663		17.0	D	72749
10.4	D	21676		17.1	D	73808
10.5	D	22193		17.1	D	73808
11.1	D	25449		17.3	D	75953
11.1	D	25449		17.5	D	78135
11.3	D	26594		17.6	D	79240
11.8	D	29588		18.1	D	84902
12.0	D	30839		18.1	D	84902
12.3	D	32774		18.3	D	87233
12.6	D	34779		18.6	D	90799
13.1	D	38279		19.3	D	99452
13.1	D	38279		20.1	D	109920
13.3	D	39735				

Plot 19:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.6	D	10007		15.8	D	60742
7.6	S	11583		15.9	D	61694
8.7	D	13963		16.0	D	62654
9.2	D	16024		16.3	D	65589
9.4	D	16896		16.3	D	65589
9.6	D	17796		16.6	D	68604
10.0	D	19679		16.8	D	70658
10.2	D	20663		17.0	D	72749
10.4	D	21676		17.0	D	72749
10.5	D	22193		17.0	D	72749
11.5	D	27769		17.0	D	72749
11.7	D	28974		17.0	D	72749
11.8	D	29588		17.3	D	75953
12.2	D	32121		17.4	D	77039
13.3	D	39735		17.4	D	77039
14.0	D	45088		17.4	D	77039
14.5	D	49160		17.5	D	78135
14.8	D	51704		17.5	D	78135
14.8	D	51704		18.1	D	84902
14.9	D	52569		18.2	D	86063
14.9	D	52569		18.2	D	86063
15.0	D	53443		18.4	D	88412
15.0	D	53443		18.7	D	92006
15.1	D	54325		19.2	D	98187
15.3	D	56115		19.4	D	100726
15.4	D	57023		19.5	D	102011
15.4	D	57023		20.0	D	108577
15.6	D	58865		20.1	D	109920
15.7	D	59799		21.0	D	122447
15.8	D	60742		22.5	D	145137

Plot 20:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.1	S	9939		15.6	D	58865
8.2	S	13743		15.7	D	59799
8.4	S	14509		15.7	D	59799
8.5	S	14900		15.7	D	59799
10.2	S	22457		16.0	D	62654
10.3	S	22955		16.0	D	62654
10.5	D	22193		16.2	D	64602
10.5	S	23970		16.5	D	67590
11.5	D	27769		16.6	D	68604
12.0	D	30839		16.6	D	68604
12.3	D	32774		16.9	D	71699
12.4	D	33434		17.1	D	73808
12.9	D	36855		17.1	D	73808
12.9	D	36855		17.2	D	74876
13.0	D	37563		17.5	D	78135
13.1	D	38279		17.7	D	80354
13.1	D	38279		18.0	D	83751
13.2	D	39003		18.2	D	86063
13.2	D	39003		18.3	D	87233
13.3	D	39735		18.5	D	89601
13.6	D	41980		18.8	D	93223
13.7	D	42744		19.0	D	95686
13.9	D	44298		19.6	D	103304
14.2	D	46692		19.8	D	105921
14.6	D	49999		19.8	D	105921
14.7	D	50848		19.8	D	105921
14.9	D	52569		19.9	D	107244
15.4	D	57023		19.9	D	107244
15.5	D	57940		20.2	D	111272
15.5	D	57940		22.6	D	146731

Plot 21:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.8	S	12280		16.4	D	66585
8.3	S	14123		16.5	D	67590
9.2	S	17804		16.5	D	67590
11.0	S	26615		16.5	D	67590
11.2	S	27717		16.8	D	70658
11.4	D	27178		16.8	D	70658
11.4	S	28843		16.8	D	70658
11.6	S	29994		16.9	D	71699
11.8	D	29588		17.0	D	72749
11.8	S	31170		17.7	D	80354
12.9	D	36855		17.8	D	81477
13.0	D	37563		18.2	D	86063
13.2	D	39003		18.2	D	86063
13.3	D	39735		18.3	D	87233
13.4	D	40475		18.4	D	88412
13.6	D	41980		18.7	D	92006
13.8	D	43517		18.9	D	94450
13.9	D	44298		19.5	D	102011
14.3	D	47506		19.5	D	102011
14.5	D	49160		19.9	D	107244
14.7	D	50848		20.0	D	108577
14.7	D	50848		20.2	D	111272
15.3	D	56115		20.2	D	111272
15.5	D	57940		20.5	D	115388
15.5	D	57940		20.5	D	115388
15.5	D	57940		20.9	D	121015
15.7	D	59799		20.9	D	121015
15.7	D	59799		22.9	D	151577
16.0	D	62654		23.4	D	159863
16.2	D	64602		24.1	D	171906

Plot 22:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.0	S	9627		18.1	D	84902
7.3	S	10580		18.1	D	84902
7.8	S	12280		18.2	D	86063
8.2	S	13743		18.3	D	87233
8.3	D	12434		18.7	D	92006
9.1	S	17372		18.7	D	92006
9.8	S	20524		18.8	D	93223
10.4	D	21676		18.8	D	93223
11.5	D	27769		19.0	D	95686
12.1	D	31476		19.1	D	96932
12.5	D	34103		19.2	D	98187
12.7	D	35463		19.4	D	100726
13.3	D	39735		19.5	D	102011
13.8	D	43517		19.6	D	103304
13.9	D	44298		19.9	D	107244
14.1	D	45886		19.9	D	107244
14.2	D	46692		20.0	D	108577
14.3	D	47506		20.2	D	111272
14.5	D	49160		20.3	D	112634
15.5	D	57940		20.4	D	114006
15.9	D	61694		20.5	D	115388
16.5	D	67590		21.1	D	123889
16.6	D	68604		21.3	D	126802
16.7	D	69626		21.4	D	128274
17.2	D	74876		21.8	D	134263
17.5	D	78135		21.9	D	135786
17.8	D	81477		22.4	D	143553
17.8	D	81477		22.4	D	143553
18.0	D	83751		22.6	D	146731
18.1	D	84902		24.0	D	170153

Plot 23:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.4	S	10909		13.9	D	44298
8.8	S	16110		13.9	D	44298
8.9	S	16524		14.0	D	45088
10.2	S	22457		14.1	D	45886
10.4	S	23460		14.2	D	46692
10.5	D	22193		14.2	D	46692
11.2	D	26018		14.4	D	48329
11.3	D	26594		14.5	D	49160
11.5	D	27769		14.5	D	49160
11.5	D	27769		14.8	D	51704
11.6	D	28368		14.8	D	51704
11.8	D	29588		14.9	D	52569
11.8	D	29588		15.0	D	53443
11.8	D	29588		15.2	D	55216
11.9	D	30210		15.3	D	56115
11.9	D	30210		15.5	D	57940
11.9	D	30210		15.7	D	59799
12.1	D	31476		15.8	D	60742
12.5	D	34103		15.9	D	61694
12.6	D	34779		16.1	D	63624
12.6	D	34779		16.7	D	69626
12.6	D	34779		17.0	D	72749
12.9	D	36855		17.2	D	74876
13.0	D	37563		17.2	D	74876
13.1	D	38279		17.3	D	75953
13.3	D	39735		17.3	D	75953
13.5	D	41223		17.4	D	77039
13.6	D	41980		18.5	D	89601
13.7	D	42744		18.5	D	89601
13.7	D	42744		18.9	D	94450

Plot 24:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.0	S	9627		15.5	D	57940
7.8	S	12280		15.7	D	59799
8.1	S	13369		15.8	D	60742
8.1	S	13369		15.8	D	60742
8.2	S	13743		15.8	D	60742
8.2	S	12068		15.8	D	60742
9.0	S	16945		15.9	D	61694
9.0	S	16945		15.9	D	61694
9.0	S	16945		16.1	D	63624
9.4	S	18687		16.1	D	63624
10.1	D	20167		16.2	D	64602
10.2	S	22457		16.5	D	67590
10.2	S	22457		16.8	D	70658
10.5	S	23970		16.8	D	70658
11.2	D	26018		16.9	D	71699
12.3	D	32774		16.9	D	71699
13.2	D	39003		16.9	D	71699
13.4	D	40475		17.0	D	72749
13.5	D	41223		17.3	D	75953
13.6	D	41980		17.7	D	80354
13.7	D	42744		17.7	D	80354
13.7	D	42744		17.9	D	82609
14.6	D	49999		18.0	D	83751
14.8	D	51704		18.0	D	83751
14.9	D	52569		18.7	D	92006
15.0	D	53443		19.2	D	98187
15.0	D	53443		19.5	D	102011
15.2	D	55216		19.9	D	107244
15.3	D	56115		20.4	D	114006
15.3	D	56115		20.5	D	115388

Plot 25:

Diameter	Position	Biomass		Diameter	Position	Biomass
7.0	S	9627		14.9	D	52569
7.1	S	9939		15.0	D	53443
7.3	S	10580		15.1	D	54325
7.3	S	10580		15.1	D	54325
7.4	S	10909		15.2	D	55216
7.6	S	11583		15.4	D	57023
7.8	S	12280		15.7	D	59799
8.1	S	13369		15.8	D	60742
8.4	S	14509		15.8	D	60742
9.3	D	16457		15.8	D	60742
10.5	D	22193		15.9	D	61694
12.1	D	31476		16.1	D	63624
12.1	D	31476		16.1	D	63624
12.6	D	34779		16.3	D	65589
12.9	D	36855		16.3	D	65589
13.1	D	38279		16.3	D	65589
13.2	D	39003		16.3	D	65589
13.3	D	39735		16.5	D	67590
13.3	D	39735		16.5	D	67590
13.4	D	40475		16.6	D	68604
13.6	D	41980		16.7	D	69626
13.8	D	43517		16.8	D	70658
14.0	D	45088		16.9	D	71699
14.1	D	45886		17.0	D	72749
14.2	D	46692		17.0	D	72749
14.2	D	46692		17.1	D	73808
14.2	D	46692		17.1	D	73808
14.8	D	51704		18.3	D	87233
14.8	D	51704		18.7	D	92006
14.8	D	51704		19.1	D	96932

CURRICULUM VITAE

ZACHARY J. BORTOLOT

CONTACT INFORMATION

Institute for Regional Analysis and Public Policy
Morehead State University
100 Lloyd-Cassity Building
Morehead, KY 40351
Telephone: (606) 783-9437
Fax: (606) 783-5092
E-mail: z.bortolot@moreheadstate.edu

EDUCATION

- 2004 Ph. D. (Forestry), Virginia Polytechnic Institute and State University. Dissertation title: An Adaptive Computer Vision Technique for Determining the Biomass of Loblolly Pine Plantations using Digital Orthophotography and LiDAR Imagery.
- 2000 M. Sc. (Forest Resources Management), University of British Columbia. Thesis title: Determining Stand Ages in a Hyperspectral Image Using Artificial Neural Networks.
- 1997 Sc. B. (Geological Sciences), Brown University. Thesis title: The Effects of Low Pressure and High Temperature on the Mid-Infrared Reflectance Spectra of Labradorite and Olivine.

EXPERIENCE

- 2003 - Present **Assistant Professor**, Institute for Regional Analysis and Public Policy, Morehead State University.
- 2000 - 2003 **Graduate Research Assistant**, Department of Forestry, Virginia Polytechnic Institute and State University. Multiple projects related to remote sensing, carbon sequestration by forests, spectroscopy, and urban forestry.
- 2000 - 2002 **Graduate Teaching Assistant**, Department of Forestry, Virginia Polytechnic Institute and State University. Assisted in courses on photogrammetry and forest ecology.
- 1999 **Sessional Lecturer**, Department of Forest Resources Management, University of British Columbia. Designed and taught fourth year courses on remote sensing and air photo interpretation.

- 1998 - 1999 **Graduate Research Assistant**, Department of Forest Resources Management, University of British Columbia. Created metadata for the department's collection of GIS data (1998) and developed and evaluated a technique for early detection of mountain pine beetle infested tree stands (1999).
- 1997 - 1999 **Graduate Teaching Assistant**, Department of Forest Resources Management, University of British Columbia. Assisted in courses on remote sensing, air photo interpretation, and land information systems.

TEACHING INTERESTS

- Remote sensing
- Air photo interpretation and photogrammetry
- Physical geography
- Forestry
- Image processing
- Computational methods

RESEARCH INTERESTS

- Remote sensing
- Spectroscopy
- Computer vision
- Forests in populated areas
- Forest carbon sequestration
- Artificial intelligence

COURSES TAUGHT

- Physical Geography (GEO 101 at Morehead State University)
- Basic Computer Techniques in Regional Analysis (RAPP 200 at Morehead State University)
- Photo Interpretation of Forest Land (FRST 442 at the University of British Columbia)
- Remote Sensing in Forestry and Agriculture (FRST 443 at the University of British Columbia)

UNDED GRANTS, CONTRACTS, AND AWARDS

- 2004 "A spectroscopic technique for determining the leaf nitrogen concentration of crop plants," Morehead State University Internal Research Grant, \$1690 (with C. B. Rogers).
- 2001 "An automated method for using air photos to determine the amount of carbon removed from the atmosphere by urban forests," International Society for Arboriculture, \$11,890 (with R. H. Wynne).

REFEREED JOURNAL ARTICLES

- 2003 Bortolot, Z. J. and R. H. Wynne. A method for determining fresh green leaf nitrogen concentrations from shortwave infrared reflectance spectra measured at the canopy level that requires no in situ nitrogen data. *International Journal of Remote Sensing* 24: 619-624.
- 2001 Bortolot, Z. J., C. A. Copenheaver, R. L. Longe and J. A. N. van Aardt. Development of a white oak chronology using live trees and a post-Civil War cabin in south-central Virginia. *Tree-Ring Research* 57: 197-203.

PROCEEDINGS PAPERS

- 2003 Bortolot, Z. J. and R. H. Wynne. An adaptive technique for automatically digitizing tree crowns in high resolution aerial images. In: *Proceedings of the ASPRS National Convention*, Anchorage, AK., May 5-9, 2003. 12pp.

NON-REFEREED PUBLICATIONS

- 1999 Bortolot, Z. J. and P.A. Murtha. 1999. A mountain pine beetle attack probability map derived from TM data. Report prepared for Slocan Plateau Forest Products, Vanderhoof, BC, Canada. 14pp plus maps.

PAPERS PRESENTED AT PROFESSIONAL MEETINGS

- 2002 Bortolot, Z. J. and R. H. Wynne. A computer vision / evolutionary algorithm technique for automatically determining tree locations and crown diameters in digital aerial photographs. IUFRO Symposium on Statistics and Information Technology in Forestry, Blacksburg, Virginia, September 8 - 12, 2002.
- 2002 Bortolot, Z. J. and R. H. Wynne. An automated algorithm for finding tree locations and diameters using high resolution normal color or color-infrared orthophotographs. ACSM-ASPRS Conference and Technical Exhibition, Washington, DC, April 19-26, 2002.

2000 Murtha, P.A., Z. J. Bortolot and J. Thurston. A Landsat TM spectral unmixing mountain pine beetle attack fraction map in the Vanderhoof Forest District, British Columbia. 22nd Canadian Remote Sensing Symposium, Ottawa, Canada.

PROFESSIONAL SERVICE ACTIVITIES

2003 Institute for Regional Analysis and Public Policy, Research and Development
Director search committee

2002 – 2003 Treasurer, Graduate Student Assembly, Virginia Polytechnic Institute and State University

2002 – 2003 Graduate Representative, Strategic Budgeting Board, Virginia Polytechnic Institute and State University

2002 – 2003 Secretary, Graduate Student Assembly Budget Board, Virginia Polytechnic Institute and State University

2001 – 2003 President, Virginia Tech Chapter, American Society for Photogrammetry and Remote Sensing

2000 – 2001 Treasurer, Virginia Tech Chapter, American Society for Photogrammetry and Remote Sensing