ForestTaxator: A Tool for Detection and Approximation of Cross-Sectional Area of Trees in a Cloud of 3D Points

Maciej Małaszek, Andrzej Zembrzuski, Krzysztof Gajowniczek
Institute of Information Technology
Warsaw University of Life Sciences – SGGW, Warsaw, Poland
krzysztof_gajowniczek@sggw.edu.pl

Abstract. In this paper we propose a novel software, named *ForestTaxator*, supporting terrestrial laser scanning data processing, which for dendrometric tree analysis can be divided into two main processes: tree detection in the point cloud and development of three-dimensional models of individual trees. The usage of genetic algorithms to solve the problem of tree detection in 3D point cloud and its cross-sectional area approximation with ellipse-based model is also presented. The detection and approximation algorithms are proposed and tested using various variants of genetic algorithms. The work proves that the genetic algorithms work very well: the obtained results are consistent with the reference data to a large extent, and the time of genetic calculations is very short. The attractiveness of the presented software is due to the fact that it provides all necessary functionalities used in the forest inventory field. The software is written in C# and runs on the .NET Core platform, which ensures its full portability between Windows, MacOS and Linux. It provides a number of interfaces thus ensuring a high level of modularity. The software and its code are made freely available.

Key words: point cloud, genetic algorithms, trees, 3D scan.

1. Introduction

A point cloud is an unordered collection of points in the R^3 space, which is in fact a three-dimensional image (most often of a real-life, scanned space). Due to the different parameters of devices scanning their surroundings, each point may contain additional information, such as the strength of light reflection or the color of a given point. Such information are especially useful when the cloud is used to create a three-dimensional model of a real object along with its colors.

Working with a 3D point cloud very often can be difficult, because it contains even billions of spatial points and for this reason, at a given time one often works with a small segment of the cloud [1]. The point cloud is often stored as a text file in ASCII format, due to ease of reading. In such a file each point is stored as three Cartesian coordinates (X, Y and Z) [2]. One effective way to manage such a cloud is to use the octal tree algorithm. Appropriate implementation makes it possible to obtain not only the faster access to a group of points located in the vicinity, but also the data compression even up to 10% of the uncompressed data size [1]. The potential uses of such an image are very wide as it provides relatively accurate information about the shape and size of a given object or environment. Examples of areas in which the point cloud is used are industrial automation, architecture, agriculture, construction and maintenance of tunnels and mines, urban space planning, as well as inventory of forest resources [1, 2].

Inventory of forest resources is usually performed using a terrestrial laser scan (TLS). The use of this technique allows for quick, effective and automatic determination of the basic parameters of the stand, i.e., the number and location of trees, their diameter at a height of about 1.3 m (the so-called *diameter at breast height*, DBH) and the shape of the trunk and crown [2]. By applying the selected algorithm of matching a circle or an ellipse to the cross-section of a tree, it is possible not only to detect the tree, but also to determine the volume of the trunk, which is an important information in forest management [1].

In this work, the problem of 3D point cloud analysis in searching for information about trees will be presented. The technical part will be focused on solving the problem of detecting and modelling a tree using genetic algorithms. Both the tree detection algorithm and the algorithm for modeling a tree with ellipses will be described. The tree detection algorithms can be interpreted as a filtering algorithm, and by detection is meant identifying the points that make up the tree trunk and discarding those points which make up branches, leaves and other objects. The tree model will consist of a collection of ellipses saved in a tree structure. The ellipses will inform about the shape of the trunk, its location and diameter. The use of ellipses instead of circles is a non-standard approach, due to the lack of a formula that would allow analytical determination of the distance of any point from the circumference of the ellipse, but but the model based on ellipses will achieve much better results than the model based on circles in the case of tilted trees.

To date, there is no commercial software for working with TLS data that covers all aspects of data processing for forestry purposes [2]. Two companies have developed such software, i.e., Treemetrics (operating in Ireland) has developed the *AutoStem* software [3], and Taxus IT (operating in Poland) has developed the *tScan* software [4]. However, in both cases, they have not yet been included in their companies' commercial offerings. Nevertheless, there are several free solutions (Table 2 in Section 8) that enable such TLS data processing, mainly developed by research centers. These include standalone desktop applications and libraries of specialized tools used in software development environments such as Matlab, R or Python. Unfortunately, none of them provides all necessary functionalities in one place.

To fill this gap, in this paper we present a novel software called *ForestTaxator* [5]. The software supports tree detection in the point cloud and the development of three-dimensional models of individual trees. The software and its code are made freely available. A library providing ready-made solutions for creating genetic algorithms is also available. This library was entirely designed, constructed and distributed under the MIT license by the author of this work and is his sole intellectual property. The main contributions of this paper can be summarized as follows.

1. It provides all necessary functionalities in one place compared to other freely available software.
2. It provides innovative noise removal filters, for both non-tree elements and noise resulting from scanner error.
3. It employs genetic algorithms to solve the complex problem of tree detection and its cross-sectional area approximation with ellipse-based modelling 3D point cloud.
4. In both the detection and approximation algorithms various version of genetic algorithms are applied and numerically tested, achieving very good accuracy in relatively short time.
5. The analysis is performed using our software, which is made freely available.

The remainder of this paper is organized as follows. Section 2 provides an overview of the similar research for detection and approximation of cross-sectional area of trees in a cloud of 3D points. In Section 3, the theoretical framework of the proposed algorithms is presented. In Sections 4 and 5 the detection and approximation algorithms are described in detail. Section 6 presents the architecture, the functionalities of the software along with the illustrative example. Section 7 outlines the numerical experiments and presents the discussion of the results. Section 8 outlines the impact of the software together with a comparison between *ForestTaxator* and other free tools. The paper ends with concluding remarks in Section 9.

2. Literature review

TLS scanners work by measuring the distance and the horizontal and vertical angles between the device and the object under examination usinglaser light beams emitted by the device [2]. Nowadays, there are several models of TLS on the market, which can be divided into two main groups: phase shift scanners and time-of-flight (ToF) scanners. The main distinguishing feature between these two scanners' types is the distance measurement technology [6]. ToF scanners measure the distance less accurately than phase-shift scanners; nevertheless, the data obtained with phase-shift scanners are subject to noise. ToF scanners tend to have a greater data measurement range compared to phase-shift scanners [7]. A main superiority of ToF scanners is the ability to record many reflections of the laser beam. This is especially important when scanning objects near vegetation [2, 6].

The TLS data acquisition in the forest environment can be performed on three different levels, i.e., sample plot, individual tree and whole stand. Sample plots are usually performed in single scan (SS) or multi scan (MS) data acquisition mode [8], where the latter one is much faster. The main disadvantage of the SS mode data acquisition is the high likelihood of the so-called 'occlusion effect' [9], i.e., some trees to be omitted because trees in the same azimuth relative to the plot center obscure each other [10].

TLS data gathered based on individual trees are particularly useful in improving or developing allometric equations for traits such as whole-tree volume or biomass [2]. The main pros is the speed, non-invasiveness, and precision of obtaining stem morphological curve information. This kind of scanning is usually done in MS mode, where scanner locations are placed around the tree from at least three positions [11]. Nevertheless, this number should be selected depending on the size of the object being scanned and the planned level of data detail [12]. The MS mode is usually employed for the whole stand scanning, but is should be remembered that this mode is usually more complicated due to the need of repositioning locations of the scanning positions [13].

When planning data acquisition with TLS, in addition to technical aspects such as determining scanning parameters and selecting the appropriate data acquisition mode, external environmental factors must be considered, such as weather conditions and vegetation period. Optimal conditions for TLS are windless days without precipitation, moderate temperatures, and low humidity [2, 14]. Wind is a factor that can significantly affect the quality of data collected (occur when the average wind speed does not exceed 5 m/s [15]), especially in the tree canopy. Scanning can be driven in light rain or fog, but is also not recommended, as well when there is snow cover [16]. For surveys designed to obtain the most accurate data on the morphology of woody parts of trees and to estimate their volume or biomass, the best time to scan is in early spring or late fall [2].

TLS data processing for dendrometric tree analysis can be divided into two main processes: detection of trees in the point cloud and creation of 3D models of individual trees [2]. The first process usually consists of two stages, where the first phase involves placing a thin horizontal slice of a 3D point cloud on a horizontal plane. During the next phase, trees are detected using clustering algorithms grouping the points and next by searching for some geometric shapes, such as circles or semicircle [17, 18]. Unfortunately, this assumption does not hold for complex stands with high tree density or undergrowth [19]. To address this issue, practitioners have developed methods for accurately identifying stems or woody parts of trees directly in the point cloud. Authors in [20] presumed that woody parts have a higher reflectance intensity (normalizing of this value is time-consuming and complicated [19]) than leaves. Authors in [23] used a different approach by iteratively employing the local geometric features of point clouds, where the nearest groups of points, defined by the radius of the sphere or the number of nearest neighbors around the focal point, are selected by the principal component analysis (PCA). Geometric features of point clouds like verticality, flatness and linearity were used for trunk detection [6, 24]. Finally, machine learning algorithms were used to classify the trunk as well [25]. The second process usually uses the quantitative structure model (QSM). Such a model is assumed to represent tree morphology as accurately as possible and is fully measurable since these properties allow for accurate determination of the thickness and volume of aboveground woody components [2]. Authors in [6] distinguish five levels of detail of digital tree models that allow for different characterizations of the modelled trees [2].

3. Theoretical framework of the proposed method

3.1. LiDAR

LiDAR is a mechanism that makes it possible to accurately record the elements of the environment. The coordinates of the points representing the individual details of the scanned objects are recorded in the Cartesian system and conventionally referred to as X, Y and Z coordinates. Each recorded point represents a point in real space from which the laser beam was reflected. The coordinates are computed from the distance measurement and the vertical and horizontal angle measurements at which the laser beam is emitted. Each LiDAR laser scanner has two degrees of freedom that describe the azimuth and elevation of the scanner at a given moment. With this information, the scanner records the reflection of the beam and can record the exact Cartesian coordinates of the detected opaque point. LiDAR scanners record data at the speed of thousands of points per second, achieving an accuracy of the order of millimeters.

There are two methods of measuring the distance. The first is the method of measuring the phase shift of the electromagnetic wave. The phase shift of the wave is proportional to the distance traveled by the beam [26] and can be expressed by the formula:

$$d = \frac{\frac{c}{2f}\phi}{2\pi}\,, \tag{1}$$

where d – distance of the reflection point from the emitter, ϕ – phase shift (delay), f – amplitude modulation frequency of the emitted beam, c – speed of light. The intensity of the wave, in other words – the brightness of the light emitted by the laser, is modulated. The length of the electromagnetic wave may differ between scanner models – there are models that emit waves in the visible light range as well as in the near infrared range. Most often, however, it is in the range from 600 nm to 800 nm. This method is characterized by high accuracy, high number of scanned points per second and an average range of effective measurements (up to about 150 m).

The assumptions of the second method are much simpler because it consists in measuring the time between sending the laser pulse and receiving its reflection, and the distance can be determined by the formula:

$$d = \frac{\frac{1}{2}c}{t}\,. \tag{2}$$

Unlike the previous method, the emission of the wave is not continuous, but is a single pulse. The implementation of these assumptions requires the use of components operating at very high frequencies, in the order of tens of GHz, and this greatly complicates the achievement of a precise device. This method allows for measurements at a maximum range of up to 2000 m.

3.2. Genetic algorithms

Genetic algorithms are a subset of evolutionary algorithms. They are inspired by natural evolution and follow its principles by implementing mechanisms of natural selection, mutation and inheritance. The idea of evolutionary algorithms has been around for almost 60 years, but its foundations remain unchanged. There is a large class of optimization problems for which no specialized algorithms have been developed. In the case of simpler problems, the solution space of which is small, iterative search can be used. In the case of huge spaces, the evolutionary solutions are usually applied. Genetic optimization is a stochastic approach in which the way of searching the solution space is modeled by genetic inheritance and Darwinian competition for survival.

The feature that distinguishes the evolutionary approach from other algorithms is its genetic resistance to the phenomena affecting the operation of traditional analytical optimization methods, i.e. the lack of of necessity to use derivatives, insensitivity to the presence of discontinuities in the solution space, or ability to step over local extrema. Of course, this does not mean that the evolutionary algorithm ensure that a globally optimal solution is found. The robustness of these algorithms results from the combination of four features: encoding the parameters, operating on populations, introducing the element of randomness and using the minimum necessary information about the problem.

As mentioned before, when describing genetic algorithms, concepts derived from natural genetics are used. The meaning of all of these concepts should therefore be clearly stated. An individual represents exactly one element in the search (or solutions) space of a given problem. A population is a collection of individuals or solutions. Fitness (or objective) function is a measure of the fit of a specific individual to the solution sought. Selection is a method of stochastic or deterministic selection of individuals which will pass on their features in the form of genetic material to the next generation. The foundations of the genetic algorithms have been described for example in [21, 22].

4. Detection algorithm

4.1. Genetically distinguishing the arrow of the tree from branches, leaves and noise

The first stage of the analysis, after cleaning the data, is to find the points shaping the tree's arrow. This can be done in various ways, including the Hough transform, but we decided to use a hybrid genetic algorithm for this. In order to solve the problem with

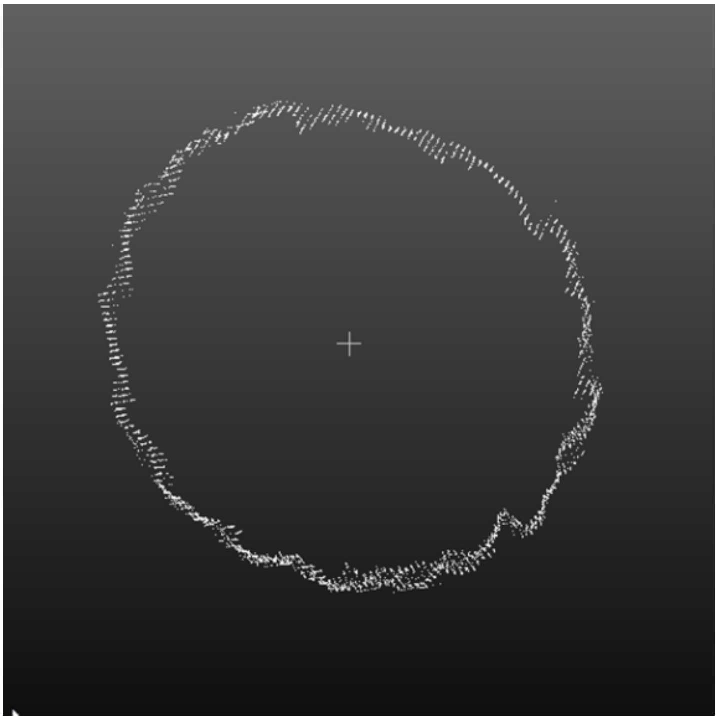
Fig. 1. Vertical projection of the cross-section of the tree scan at a height of about 1.5 m.

the help of an evolutionary algorithm, a simplified model is needed. Once a model is identified, an evaluation function should be designed that clearly identifies which of the two potential solutions is better. It is also necessary to define an encoding method that will make it possible to use the selected crossover operators.

4.2. The perfect arrow model

As has been said before, the cross section of the tree arrow can be approximated with a circle. The data, from the RemBioFor project [28], were made in a multi-station mode, i.e. from three stations located at the vertices of the triangle. As a result, most of the scanned tree lacks places shaded by the tree itself. This is illustrated in Figure 1.

The density of the points remains approximately the same. Therefore, in a circle there would be a relationship in which the distribution of points in any chosen axis would resemble a symmetrical bimodal distribution, which would be similar to the sketch presented in Figure 2.

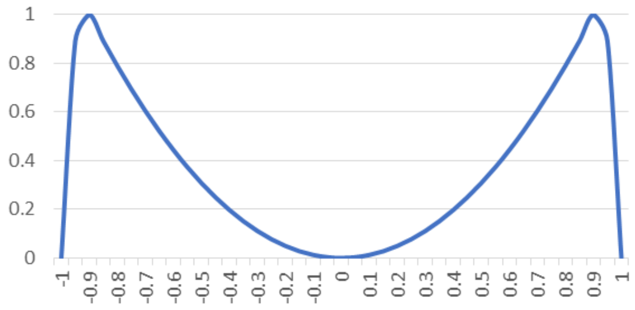

Fig. 2. Sketch of a 3D point distribution for a circle.

Since the distribution will be identical regardless of the selected axis, one can select coordinate axes lying in the plane of the cross-section. Thanks to this, it will be possible to precisely define the shape of the points. The exact dimensions of the point group are also known, which makes it possible to normalize the domain of the function that will be used to approximate the distribution. To calculate the distribution, the entire group should be divided into equal fragments, and then it should be counted how many points are there in the given fragment of the group. The denser the split, the smoother the distribution will be, but at the same time there will be larger deviations due to the structure of the tree. In our solution, we decided to divide the groups into 32 equal parts. For a tree with a diameter of 50 cm, the width of one group will be 1.5625 cm. In addition, we decided to omit the first two and last two fragments in the analysis. This allows us to approximate the distribution using the quadratic function.

It should be emphasized that the above simplified approach is used only to detect trees, while the shape and size of the cross-section of a detected tree will be later much more accurately approximated using ellipses.

In summary, in the ideal tree arrow model it is assumed that it is a circle with an even distribution of points on the tree surface. This model is represented by two bimodal distributions that will be approximated by quadratic functions. The parameters of the entire model are the parameters of both square functions, i.e. six floating point values. This is a case in which the application of genetic algorithms is simple and at the same time can give very good results.

4.3. Fitness function

The selection of the evaluation function should reflect the properties of the analyzed problem. The function needs not be differentiable, but should be continuous as far as possible in the domain of individuals that may arise. The problem of discontinuity could be avoided, for example, by introducing the lethality of some mutations, but this would introduce an additional performance overhead.

In the adopted model, the fitness function is the similarity of the distribution of points in perpendicular axes to the quadratic function. Similarity is best determined by calculating the differences for each of the 28 fragments. The requirement for the fitness function is that it must return one floating point number, which forces additional operations on the calculated differences. A frequently used aggregation function is the arithmetic mean, but it is not immune to outliers. Also, when some values are greater than the model values, and others are smaller, the average difference may be 0, and in fact the error will be large and the analyzed group of points will not be a circle. Therefore, it is also worth using the standard deviation, which measures how far individual values are from the mean value.

Both metrics should be used to form a single number, ideally this number should be 0. We have arbitrarily chosen the sum of the absolute values of the mean and the standard deviation. Thus, the following model arises:

$$\mathrm{avg}_i = \frac{\sum_{x=3}^{30} \mathrm{diff}_i[x]}{28}, \tag{3}$$

$$\mathrm{stdDev}_i = \sqrt{\frac{\sum_{x=3}^{30} (\mathrm{diff}_i[x] - \mathrm{avg}_i)^2}{28}}, \tag{4}$$

$$f_i = |\mathrm{avg}_i| + |\mathrm{stdDev}_i|, \tag{5}$$

$$f = \max_i f_i, \tag{6}$$

where $\mathrm{diff}_i[x]$ is the difference between the model and the individual for the fragment with index x in the i-th axis, avg_i is the mean value of the difference for the i-th axis, stdDev_i is the population standard deviation for the i-th axis, f_i is the value of the fitness function on the i-th axis, and f is the value of the fitness function for the individual.

The above function will make it possible to clearly define which individual is closer to the model and which is farther.

4.4. Genetic representation

The quadratic function will be used in the general form: $y = ax^2 + bx + c$. It is known that the second derivative must be positive. It follows that the parameter a must be positive. It is also known that the number of points in the distribution will not be lower than 0, so parameter c must also be non-negative. The model takes into account normalization of the values so that the values are in the range $[0; 1]$. This gives information in what interval the values of the quadratic function should be, and indirectly – in what interval the parameter values should be. If the maximum value of the function exceeds 1.5, it will be known that the given group is noise. Therefore, it can be assumed that the range within which the parameter a will change should be equal to the range $[0.01; 1.5]$. The remaining parameters should oscillate close to zero, but in order to introduce some flexibility we chose the interval $[0; 0.5]$.

The scope of parameter changes has been limited. This eliminates the need to use larger variable types such as `double` or `int32`. Our solution assumes operating on an array of six `unsigned short` variables, i.e. 16 bit integers. This will give 65 536 possible values for each parameter. For the interval $[0.01; 1.5]$ it gives the realtive accuracy of 0.00002.

The solution will therefore be represented as a 96-byte string. The `CollectiveGenotype` class from the GeneticToolkit library [30] will be used for this purpose. As a parameterizing type, a special class `ParabolicParameters` will be used, containing a total of 6 floating-point variables (3 for each axis) as well as serializing and deserializing functions. The representation introduces no domain restrictions, so there is no need for a lethal mutation mechanism.

5. Approximation algorithm

5.1. Model for the approximating algorithm

Most often, circles are used to approximate the cross-section of a tree. It may happen that the cross-section of the tree is closer to the ellipse. The most important reason for this is that the tree can grow at a certain angle that is not taken into account when splitting the point cloud into layers. Therefore, in this work, we will use an ellipse as a model.

The ellipse-based model has the benefit of being able to obtain better results in the fitness function compared to the circular-based model. A significant disadvantage of this solution is the introduction of a greater number of degrees of freedom, which increase the computational complexity of the problem. In a two-dimensional space, a circle has three degrees of freedom: two center coordinates and a radius. An ellipse in the same space has five degrees of freedom: two coordinates of the first focus, two coordinates of the second focus, and the length of the major semi-axis. The model we have adopted will consist of three variables:

1. Two-dimensional coordinates of the first focus.
2. Two-dimensional coordinates of the second focus.
3. Length of the great driveshaft.

5.2. Fitness function

The fitness function will take into account the distance of points from the set under consideration to the ellipse. The distance of a point from the ellipse is meant as the Euclidean distance of a point in two-dimensional space from the nearest point on the circumference of the ellipse. If we designate the focal points of the ellipse as F_1 and F_2, and any point lying on its circumference as P, then the major axis marked as a has the following relationship:

$$|PF_1| + |PF_2| = 2a\,. \tag{7}$$

Unfortunately, calculating the distance of any point from the circumference of the ellipse is not a trivial task, let alone a task with low computational complexity. However, as mentioned earlier, the evaluation function is primarily used to determine which of the two individuals is better, that is, closer to the expected solution. For this reason, we decided to introduce the following relationship for any point P:

$$d \sim |PF_1| + |PF_2| - 2a. \tag{8}$$

Such a relationship is not linear – the same increase in the distance of the point to the ellipse results in a different increase in the value of d depending on how far P was before moving away. This is advantageous because the farthest points will penalize the subject much greater than points close to the ellipse. As a result, minor distortions, caused for example by the shape of the cortex, will not have as large an effect as if the distance between the point and the ellipse were correctly measured. Eventually, we are going to use the following value as the fitness function:

$$f = \sum_i \frac{|P_iF_1| + |P_iF_2| - 2a}{n}\,, \tag{9}$$

where n stands for the number of points.

5.3. Genetic representation

The genotype must encode five floating point variables. The coordinates of the foci can basically have any values, therefore it is impossible to narrow down the range in which they should be searched for. In the case of the large driveshaft, it is known that it must be a value greater than 5 cm, and it will also be much smaller than 5 m. For all variables, differences not exceeding 0.01 mm will not seriously affect the results. Assuming a very rough approximation that the tree is a cylinder 30 m high, the accuracy in the calculated volume will be about 0.000000942 m^3. This is a completely negligible error, and adopting such precision will allow for the use of single precision floating-point numbers.

The above assumptions indicate that it will be efficient to use the `CollectiveGenotype` class with the `EllipticParameters` parameterizing type. The `EllipticParameters` class will work analogously to the `ParabolicParameters` class, that is, it will contain five floating point variables as well as serialization and deserialization functions, taking into account the assumptions made above regarding the search ranges.

6. Software Description

6.1. Software architecture

The software is written in C# and runs on the .NET Core platform, which ensures its full portability between Windows, MacOS and Linux. It provides a number of interfaces thus ensuring a high level of modularity. The software development is supported by the use of the Azure CI/CD (Continous Integration – Continous Delivery) system, which allows to design the procedure for testing and implementing the solution. The quality of the code from the beginning of the project has been maintained at the highest A level in terms of ease of maintenance, reliability and security. In order to work efficiently the *ForestTaxator* requires some 3D point cloud and mesh processing software such as *CloudCompare* [31].

The software is available at the GitHub repository and after downloading the source files it can be compiled using the following command:

```
dotnet build -c Release forest-taxator
```

ForestTaxator provides all functionalities in user friendly console application, i.e. it takes input and displays output at a command line console with access to three basic data streams: standard input, standard output and standard error. By calling the following command

```
./ForestTaxator.Application --help
```

the user sees that the software is composed of two main functionalities.

```
ForestTaxator.Application 1.0.0
Copyright (C) 2021 ForestTaxator.Application
analyze
convert
help      Display more information on a specific command.
version   Display version information.
```

The convert functionality allows to transform one file structure into another, such as, PCD (Point Cloud Data) file into XYZ or GPD (Generic Printer Description) file into XYZ. The analyze functionality aggregates all necessary functions allowing to conduct the entire study step by step.

```
ForestTaxator.Application 1.0.0
Copyright (C) 2021 ForestTaxator.Application
approximate-trees Approximates tree trunk using ellipses
detect-trees
filter
slice
```

```
terrain
tree-height
help            Display more information on a specific command.
version         Display version information.
```

6.2. Illustrative example

As an input ForestTaxator accepts flat files such as .txt, .csv or binary compressed PCD file format. In this section we present an illustrative example performed on 3D Forest dataset [32] stored as a flat files. Dataset consists of multiple layers constituting the base cloud where all points of plot are present. Below we present all necessary commands along with the required parameters (and their short description).

1. Extract terrain height map from `example.xyz` file into `terrain.thf`.

   ```
   Input: Source point cloud file
   -o -> path & filename where output file has to be saved
   ./ForestTaxator.Application analyze example.xyz
   -o output/terrain
   ```

2. Extract tree height map from `example.xyz` file into tree-height – it takes highest point from each square in a grid.

   ```
   Input: Source point cloud file
   -o -> path & filename where output file has to be saved
   ./ForestTaxator.Application analyze tree-height example.xyz
   -o output/tree-height
   ```

3. Split `example.xyz` file into collection of slices and save this collection to format that supports grouping, -t parameter specifies path to terrain height map, which is used to normalize height of each point in cloud.

   ```
   Input: Source point cloud file; Terrain heightmap
   -t / --terrain -> Path to terrain heightmap file
   -o -> path & filename where output file has to be saved
   ./ForestTaxator.Application analyze slice example.xyz
   -t output/terrain -o output/sliced
   ```

4. Create small groups of point based on their neighborhood – points that are located near each other form a group of points. Based on properties of each group, application filters out those groups that are considered to be *noise* (leaves, small branches etc.). Result is stored in filtered directory. All necessary parameters are stored and defined by the user in the configuration `.json` file (please see Listing 1 for the structure of the file).

```
Input: Sliced point cloud from slice step; Terrain height map
-c -> Path to FiltersConfiguration.json file
-o -> path & filename where output file has to be saved
./ForestTaxator.Application analyze filter output/sliced
-c Configs/FiltersConfiguration.json -o output/filtered.gpd
```

5. Convert `gdp` file into `XYZ` file.

```
./ForestTaxator.Application convert -i GPD
-o XYZ output/filtered.gpd output/filtered.xyz
```

6. Create collection of point groups. Each collection is considered to be a tree candidate. Collections are created based on XY location for each slice. Tree candidates that do not meet requirements are removed, and only remaining ones are exported to detected directory.

```
Input: GPD file from filter step of collection of XYZ files
from detect-trees step
--export-preview -> for each exported GPD file, XYZ file is created
so it can be viewed in 3d cloud viewer
-o -> path to DIRECTORY where output file has to be saved
./ForestTaxator.Application analyze detect-trees output/filtered.gpd
-o output/detected --export-preview
```

7. Create approximation of tree trunk using genetic algorithm.

```
Input: GPD file from detect-trees step; Tree height heightmap
--export-preview -> exports collection of approximating ellipses
into XYZ file. 4th value in point represents error equal to mean
distance between ellipse border and cloud point
--smooth - Smooth tree using regression on all nodes. It replaces
ellipses with circles
-h / --node-height -> float number representing height of single
slice. Required if file was sliced with different height than
default 0.1 m
-t / --tree-heightmap -> Path to tree heightmap file
-o -> path to DIRECTORY where output file has to be saved
./ForestTaxator.Application analyze approximate-trees output/detected
-c Configs/ApproximationConfiguration.json -o output/approx
-t output/tree-height --export-preview
```

Listing 1. Structure of the configuration file `FiltersConfiguration.json`.

```
{
  "LargeGroupsFilter": {
    "Order": 1,
    "LargeGroupsMaxSize": {
      "variables": [
        {
          "name": "height",
          "type": "System.Double",
          "order": 1
        }
      ],
      "expression": "Max(0.1f,␣0.75f␣-␣0.01f␣*␣height)"
    }
  },
  "SmallGroupsFilter": null,
  "AspectRatioFilter": null,
  "EllipsisMatchFilterConfiguration": {
    "FitnessThreshold": 0.1,
    "MatchEccentricityThreshold": 0.80,
    "BufferWidth": 0.002,
    "InvalidEccentricityThreshold": 0.85,
    "GeneticAlgorithmConfigurationFile":
      "Configs/DetectionEllipsisMatch.GeneticAlgorithm.json",
    "Order": 5
  },
  "GeneticDistributionFilterConfiguration": {
    "GeneticAlgorithmConfigurationFile":
      "Configs/Distribution.GeneticAlgorithm.json",
    "DistributionResolution": 32,
    "TrunkThreshold": {
      "variables": [
        {
          "name": "height",
          "type": "System.Double",
          "order": 1
        }
      ],
      "expression": "0.3"
    },
    "Order": 3
  }
}
```

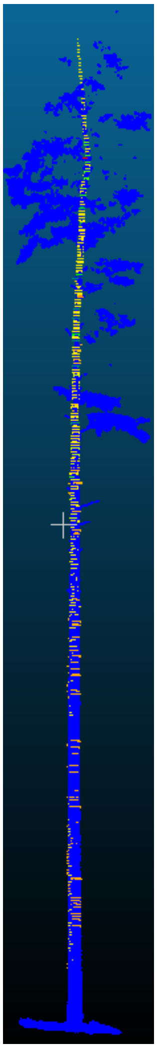

Fig. 3. Final output presented by CloudCompare.

After calling application in the last 7th step *ForestTaxator* creates `XYZ` file and `json` file. Both files consist of a collection of approximating ellipses (or circles) in a given format. After combining aforementioned `XYZ` file with the input `example.xyz` file, the final output is presented in the Figure 3, where blue dots denote the tree while yellow dotes represent the collection of ellipses.

7. Research framework and results

7.1. Experiment design

The data-set used in this article was provided by the Department of Geomatics of the Forest Research Institute with headquarters in Sękocin Stary as part of the RemBioFor project [28]. The test data includes scans of seven Scots pine trees, i.e. Hajnówka, Jedwabno, Nowe Ramuki, R_ilaw, R_lidz, Wejherowo, Wipsowo. All scans were made in a multi-station mode with the use of three stations in triangular layout. The data was manually trimmed by the employees of the Institute so that the cloud contained only the model tree. The cloud has been normalized to so that the lowest point is at height 0. All scans were made with the Faro Focus3D scanner.

Despite the high quality of the data, the scans do not show all trees due to auto-occlusion. The scans contain a lot of noise caused by branches and leaves; also, there are discontinuities in the tree trunks. This has a negative effect on the functionality of the algorithms. Halfway up the tree one can compensate for gaps by applying linear regression on correctly approximated fragments. However, the most important problem is the misrepresentation of the actual height information on the trees. This information is critical, mainly because in the formula by which it is calculated the diameters of the tree at different heights are used. In addition, a distortion in the height of the tree leads to a large error in estimating the volume of wood that can be harvested from a given single tree.

All simulations were repeated 10 times. After 10 repetitions, the mean and standard deviation were calculated. To make it justifiable to aggregate the results for all trees (comparing the results for all trees simultaneously), the results cannot be given depending on the height at which the analyzed group is located. Instead, the groups, the tree slices, were grouped by height as a percentage. This means that the height of each group was divided by the total height of the tree and multiplied by 100. This makes sense because each tree reaches a different height, so its crown begins at a different height, which strongly influences the results. Moreover, a different height could cause the last rows of the table to contain information obtained from only one tree, and the previous rows would have a very large standard deviation, precisely because of differences in the crown boundary.

In the case of the detection algorithm, the most important element of the analysis is the error matrix and the three derived measures: sensitivity, specificity, precision and negative predictive value. As previously stated, large gaps in the obtained tree trunk slices can be easily compensated by linear regression. It is therefore essential to obtain as high specificity as possible in order to filter out the noise. All tested groups were manually divided into noise and trunk fragments. The program had access to this information to generate the error matrix. The stop conditions were as follows:

1. Sufficient solution was found (fitness function value 0.22 or less).
2. Calculation time elapsed (2 s per group).
3. Maximum level of genetic homogeneity set at 90% is exceeded.

In the case of the approximating algorithm, the most important thing is to compare the results and the data on the cross-section of the tree. For this reason, the results will contain tree dimensions (defined every 1 m of height) and the corresponding values of the model will be approximated by genetic algorithms (and linear regression). The results may differ from the values indicated in the test data because the measurements were made by the authors of the test data so that the axes of the ellipses coincide with the north-south and east-west directions. It was not possible to calculate the direction in our program. It can be assumed that a correct fit will force all algorithms to indicate very similar values. The alignment results are also presented graphically by point clouds representing ellipses created by the best genetic algorithm (with the closest results). All algorithms should give very similar results because maximum available information about the problem is used. We predict that significant errors may occur in data containing trees whose cross-section is irregular in shape. Especially the "Hajnowka" set, which was prepared with an error consisting in incorrect combination of partial scans. The stop conditions were as follows:

1. Sufficient solution was found (fitness function value 0.015 or less).
2. Calculation time elapsed (1 s per group).
3. The maximum level of genetic homogeneity set at 90% is exceeded.

It is worth noting that for a genetic algorithm these are very extreme working conditions. The computation time of the genetic algorithm can be counted even in hours for complex problems. Moreover, the determination of the value of the fitness function of a sufficiently fit individual enables the quality of the obtained results to be controlled. Decreasing this value will be connected with the extension of the computation time. Without defining the maximum computation time and imposing an exact match, the algorithm could get stuck trying to find a perfectly matched individual when it might be impossible. An error of less than 1.5 cm seems to be a good compromise for the experimental conditions.

The experiments will cover selection methods, crossing methods, mutation methods and mutation policies. Among the selection methods, these will be:

1. Roulette wheel selection.
2. Ranked roulette with the mapping i_i^3.
3. A tournament with a size of 1% of the population, but not less than 2 individuals.

Crossbreeding methods:

1. Arithmetic crossover (2 parents, 2 children, 1 byte resolution).
2. One-point crossing (2 parents, 2 children).
3. Multipoint crossing (2 parents, 2 children, 3 crossing points).

4. Uniform crossing (2 parents, 2 children).

   Mutation methods:

1. Arithmetic mutation (range $[-10; 10]$ in 1 byte resolution).
2. Negation of bits.

   Mutation Policies:

1. Simple mutation (3% probability of mutation and maximum 10% mutated genes).
2. Backa method (maximum 10% mutated genes).
3. Hesser-Manner method ($\alpha = 1$, $\beta = 1$, maximum 10% of mutated genes).

7.2. Determination of the value of the parameter ϵ

The analysis was carried out with the accuracy of a single group of points. Each group was assigned the following information: a) the height of the point group; b) the value of the fitness function; c) the classification according to the threshold $\epsilon = 0.22$; d) the manual classification using CloudCompare; e) whether the algorithm made a mistake; f) does the group belong to the tree trunk (not noise). On their basis, we counted the number of errors to determine the approximate effectiveness of the algorithm, and also determined two values: a) the smallest fitness function value achieved for the noise group; and b) the highest value of the fitness function achieved for the group being a fragment of the trunk. We used these two values to calculate the mean value, we got the values presented in Table 1.

From the above table, we finally got an average value of $\epsilon = 0.24$ with a standard deviation of 0.02. The relatively low standard deviation led to the conclusion that this value would give the best results. It is a threshold that is equidistant from the values for the groups forming the trunk and from the noise groups.

Noteworthy are the graphs (in Supplementary Materials [29]) illustrating the values of the fitness function depending on the height for individual trees. The gray line in the graphs described as "EPSILON" represents the threshold value previously calculated as the mean value between the noise group with the lowest score and the trunk group with

Tab. 1. Mean value of ϵ for each dataset.

Dataset	Mean value of ϵ
Hajnówka	0.286
Jedwabno	0.233
Nowe Ramuki	0.235
R_ilaw	0.217
R_lidz	0.245
Wejherowo	0.201
Wipsowo	0.237

the highest score. When analyzing the above graphs, it can be seen that in five out of seven cases, the sets of the trunk-forming and noise groups are approximately linearly separable.

7.3. Analysis of results of the detection algorithm

A summary of all results is presented in the Supplementary Materials (in the file `DetailedResults.xlsx` [29]). A satisfactory level of specificity was achieved in each of the 72 cases, i.e. the percentage of noise groups that were rejected. The specificity of the algorithms ranges from 96.57% to 98.95%, so the difference between the best and worst algorithms is less than 2.4 percentage points (sheet `Detection_Results` in `DetailedResults.xlsx`.).

A much more pronounced difference is for sensitivity, which should be interpreted as the percentage of groups that form the trunk that are not removed by the algorithm. The worst algorithm, which turned out to be a combination of ranked roulette, arithmetic cross, arithmetic mutation, and the Hesser-Manner mutation policy, was only 67.19%. For comparison, the best algorithm, the classic two-player tournament, one-point crossover at the bit level and a constant probability of mutation involving the negation of a random bit, achieved a sensitivity of 84.43% with a specificity of 96.58%. This gives almost the lowest precision in understanding the error matrix, but at the same time the highest negative predictive value. Precision should be interpreted as the probability that the groups classified as constituting the trunk actually constitute the trunk. Likewise, a negative predictive value represents the probability that only noise is discarded.

Of course, in the case of the worst algorithm, more than 2/3 of the information about the tree has been preserved, which allows quite effective use of linear regression on the basis of groups that are correctly approximated at the approximation stage. However, low sensitivity is also associated with a higher probability of losing information about higher parts of the tree, which are very important for obtaining a reliable tree model – without them, linear regression may be biased with greater errors.

The analysis of the computation time shows some valuable facts. First, the mean standard deviation for the algorithms does not exceed 0.1 s, so it is reasonable to draw conclusions from the attached data. Secondly, the selection of the algorithm can significantly affect the results with a limited time – the best algorithm is able to remove noise from the tree scan on average in 0.12 s, and the worst one takes as long as 20.37 s for the same task. One can also see a very clear trend. Back's mutation policy causes even a hundredfold increase in computation time in relation to the constant mutation probability. The second important fact is that the fastest algorithms are those that use value roulette, and the slowest are the tournaments. This is due to the optimization of roulette calculations and the high computational complexity of the evaluation function. Value roulette optimization is based on recording the value of the evaluation function of

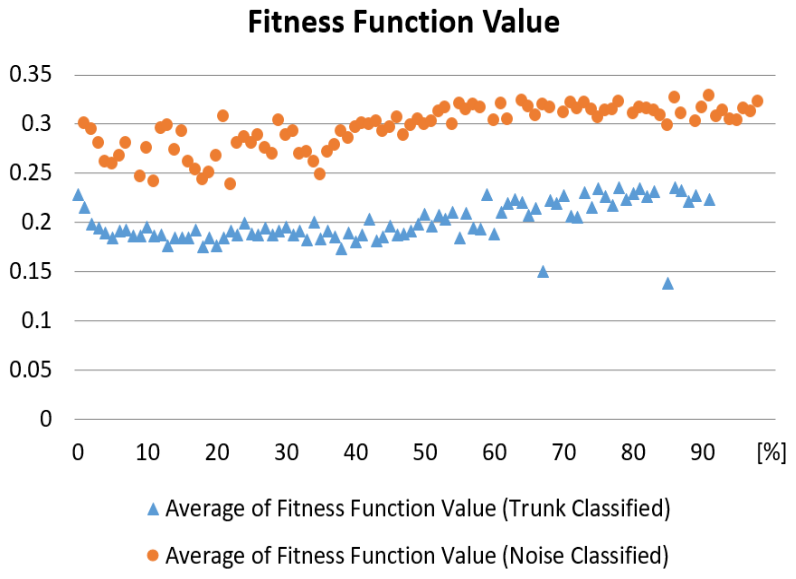

Fig. 4. Results of the best chosen algorithm called `Tournment-SinglePointCrossover-ArithmeticMutation-HesserMannerMutation-0`.

all elements every generation. As a result, the evaluation of the value of the evaluation function takes place exactly once per individual in a given generation.

Taking into account both factors – negative predictive value and average computation time – the most advantageous algorithm can be selected (`DetailedResults.xlsx` [29]). The quotient of the negative predictive value and the average calculation time shows the real profitability of using a given algorithm in a limited time. It turns out that the most effective way is to use a tournament with a single or even crossing. The method of mutation and the policy do not have much influence (except for the previously described Back mutation policy). The use of a tournament or ranked roulette is almost seven times more profitable than the use of value roulette.

The Figure 4 shows the values at which the fitness function is maintained. The difference between the groups classified as trunk and noise is large and remains stable regardless of height. The adopted threshold value of $\epsilon = 0.236214$ works very well at any height. However, it can be noticed that after exceeding 60% of the tree height, the values start to increase slightly. Increasing the threshold value to 0.25 from 60% of the tree height could positively affect the sensitivity of the algorithm. However, the current results are so satisfactory that no such test was performed.

This stage of the experiments was a definite success. The analysis showed that the noise filtering efficiency exceeds 95% while maintaining up to 84% information about the trunk. Taking into account the operating time of the algorithms, one can select one algorithm that is the fastest and the most effective and should be used as the basic one in this task. This algorithm uses a two-player tournament, binary single point crossing with an arithmetic mutation and a Hesser-Manner crossing policy.

7.4. Approximation algorithm

Due to the very low performance of Back's algorithm, we decided to exclude it from further testing. For this reason, 48 combinations will be compared instead of 72.

Analyzing results with an accuracy of 1 m of the tree does not make sense, because in practical applications the accuracy of estimating the volume of the tree as a whole matters. Therefore, we decided to present only generalized results. Moreover, in the analysis we did not perform approximation for the trunk height below or equal 1 m. Calculation close to the ground can cause large errors due to the influence of roots. The most important information is the measurement of DBH, i.e. the diameter of the tree at a height of 1.3 m, which is included in the analysis. In Fig. 5 we are presenting graphs for the `Tournament-UniformCrossover-ArithmeticMutation-HesserMannerMutation-0` algorithm from *ForestTaxator*.

As can be seen in the charts, the error usually does not exceed 5 cm. The graphs show a tendency to overestimate the trunk diameter in the lower parts of the tree and underestimate it in the higher parts. This is directly due to the dendrometric formula used in linear regression:

$$y^2 = px^r \,, \tag{10}$$

where y is the diameter of the tree, x is the distance from the top of the tree, p and r are parameters whose values vary from tree to tree. The formula in the above form is not linear, but it can be reduced to a linear form by taking a logarithm of the sides:

$$2\ln(y) = r\ln(x) + \ln(p) \,. \tag{11}$$

Since the value of x depends very much on knowledge of the tree's height, failure to do so results in an error in which the tree narrows at a different rate than it should. The lack of information about the height of the tree is inevitable with a laser scan. For this reason, this type of error will occur in any technique that uses this pattern.

It is also worth paying attention to the jump that occurs at the turn of 19 and 20 meters in the "Wejherowo" dataset. Such a jump from a negative value to a positive value means that the algorithm did not have even a single point above the height of 19.5 m that would allow to approximate the height of the tree more precisely and the diameter of the tree was approximated to the height of 19.5 m only. For the height above, 0 is

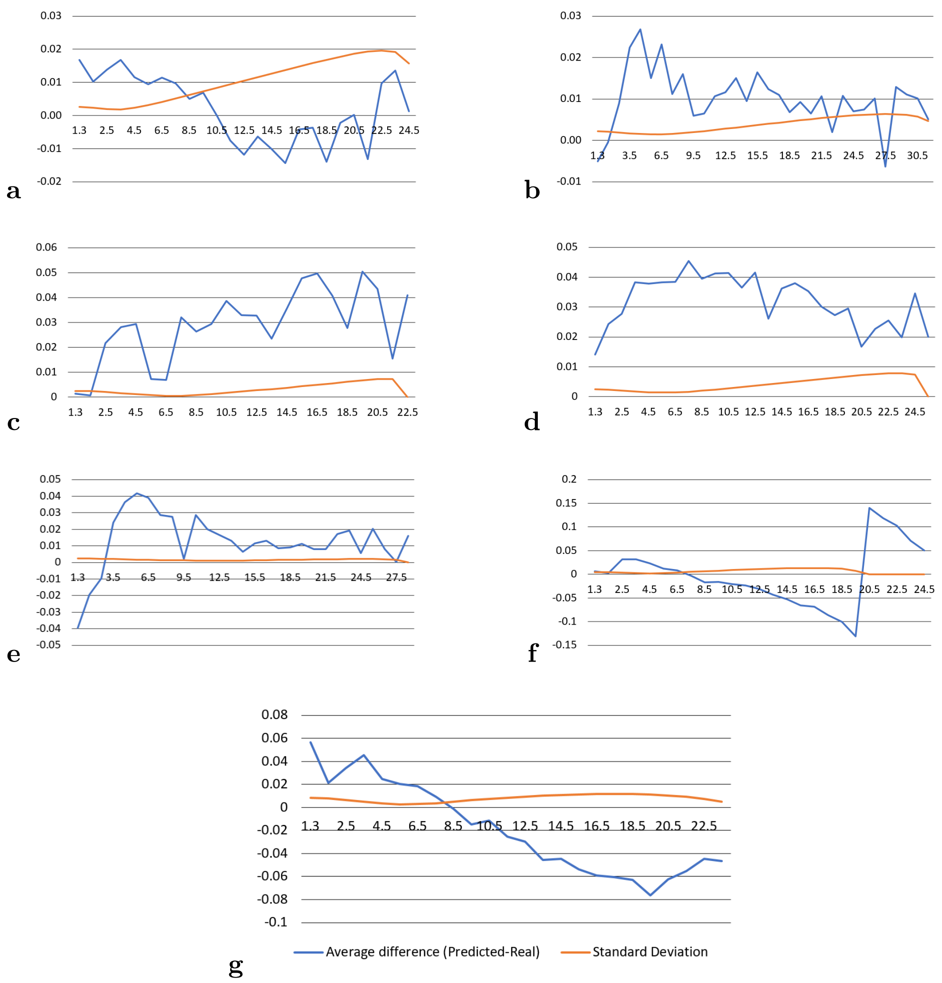

Fig. 5. Average estimation error graphs for datasets: (**a**) "Hajnówka", (**b**) "Jedwabno", (**c**) "Nowe Ramuki", (**d**) "R_ilaw", (**e**) "R_lidz", (**f**) "Wejherowo", (**g**) "Wipsowo". Legend shown below Fig. **g**.

returned. Positive values are therefore the tree diameter values that were given in the reference data.

When delving into the comparison of the results of the algorithms with the division into the measurement height, it can be noticed that the standard deviation for all algorithms does not exceed 2 cm and rather remains below 1 cm. This is illustrated in the table in `DetailedResults.xlsx`. As can be seen, the standard deviation for all heights, except for the highest parts of trees, remains within a single millimeter, regardless of the algorithm used. This shows that, in fact, none of the tested parameters significantly affects the results obtained, which can be described as very good, considering that the error rarely exceeds the value of 1.5 cm, which is a value defined as good enough.

The final comparison of the algorithms takes into account the average results for all trees as a whole. These results are summarized in the table in the Supplementary Materials `DetailedResults.xlsx`. The table clearly shows that for the effectiveness of this algorithm, the choice of the selection, crossing and mutation methods used is irrelevant. The best algorithm achieves an average error of 1.23 cm, while the worst one does so with an error of 1.68 cm. Of course, these values differ by almost 0.5 cm, but it should be noted that the standard deviation for the error also reaches values close to 1.5 cm, so more than 3 times the difference between the best and worst algorithm.

In Fig. 6 there are pictures containing ellipses representing a given tree superimposed on the point cloud with the original data. These ellipses have been marked in blue and the points have been enlarged in CloudCompare to make them easier to see.

7.5. Discussion

Several conclusions can be drawn from the conducted experiments. The experiments carried out on the detection algorithm show that in some cases the choice of crossing and selection methods can significantly affect the quality of the obtained solution. The classic solution in the form of tournament selection with two individuals, one-point bit crossing combined with arithmetic mutation and the Hesser-Manner crossing policy gave the best results in the shortest time.

The tree detection experiment also showed a significant weakness in Back's mutation policy. Its performance is very dependent on the computational complexity of the fitness function, or at best, of the individuals comparison function. Each determination of the mutation probability requires finding the best individual in the population, which involves additional operations and a high time cost.

In the case of the second experiment, one of the most important features of genetic algorithms was shown – their resistance to deficiencies and errors in the input data. Despite the lack of several meters of tree in one set and errors in the preparation of another set, the algorithm did very well, reaching an accuracy of 2 cm. An important fact is that in this case the type of selection, crossing and mutation methods used did not have any significant impact on the quality of the solutions obtained. The differences were further diminished by the operation of linear regression, which completed the missing information and smoothed the entire tree model.

In both experiments, the genetic algorithms worked very well, achieving great results and satisfactory operation time.

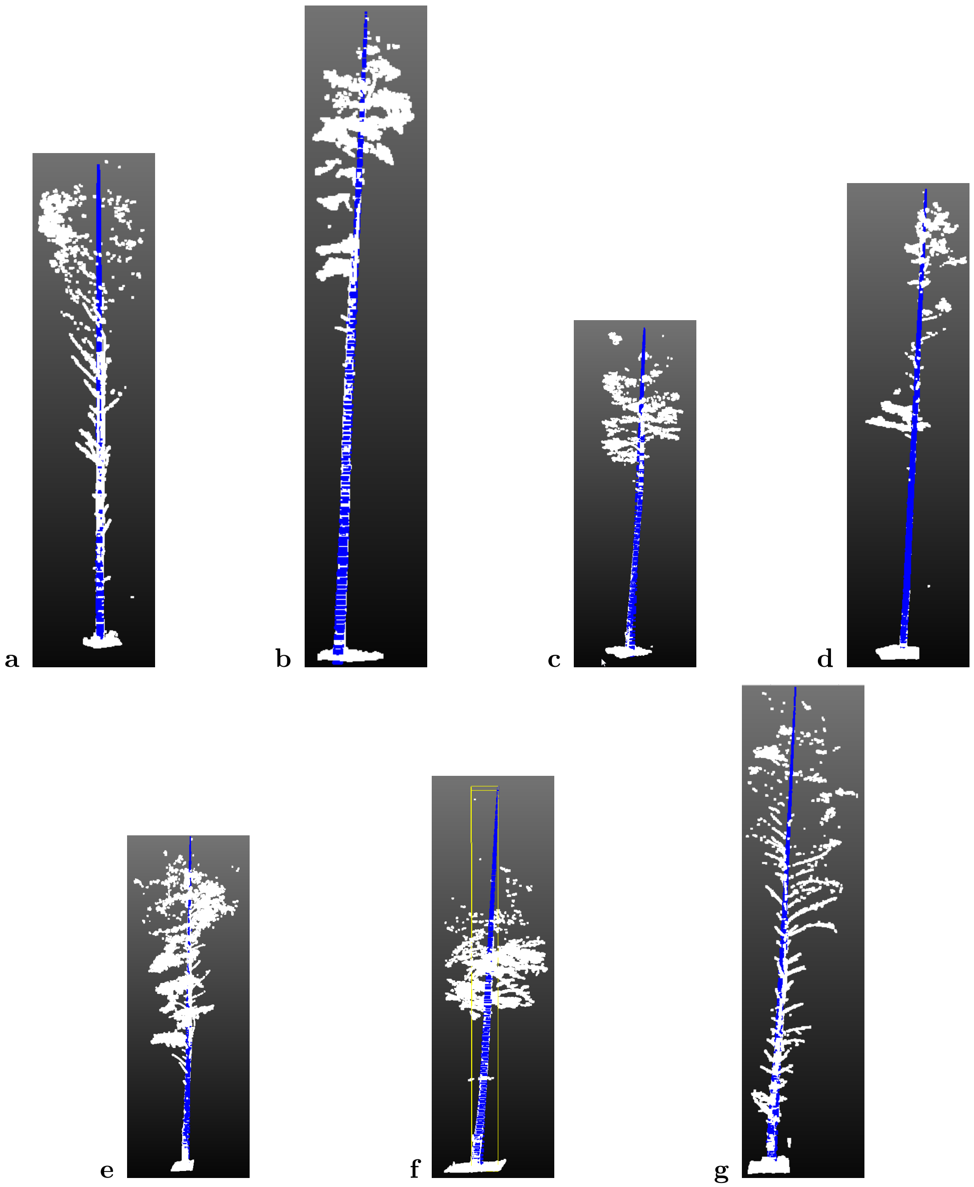

Fig. 6. Average estimation error graphs for the datasets: (**a**) “Hajnówka”, (**b**) “Jedwabno”, (**c**) “Nowe Ramuki”, (**d**) “R_ilaw”, (**e**) “R_lidz”, (**f**) “Wejherowo”, (**g**) “Wipsowo”. See text for explanations.

8. Impact

ForestTaxator represents a novel software for the forest inventory. With its publication, *ForestTaxator* becomes freely available to the scientific community and business practitioners. The impact of the software on the community is significant. It addresses the gap in the existing software. For the benefit of users the Table 2 summarizes a comparison between *ForestTaxator* and other free software. In general, there are five main functionalities that are important in this field.

First is a digital elevation model (DEM) and digital terrain model (DTM). The digital elevation model is a 3D representation of the terrain elevations found on the earth's surface. DEMs are generated from variably-spaced Lidar ground points, or they can be created using a raster grid. The DTM is a DEM in which terrain data have been further enhanced by including vector features of the natural terrain, such as rivers and ridges, providing for greater accuracy as it contains additional information defining the terrain in areas where Lidar data alone are insufficient to do the job effectively.

The second functionality allows to detect the stem of the tree. Third functionality classifies the point cloud into woody and soft parts, i.e., leaves. Forth functionality provides tree parameters, such as: diameter at breast height, tree height, tree taper curve or tree volume.

Finally, our software reconstructs the quantitative structure models (QSMs) of trees from point clouds. A QSM consists of a hierarchical collection of cylinders which estimate topological, geometrical and volumetric details of the woody structure of the tree. The input point cloud, which is usually produced by a terrestrial laser scanner, must contain only one tree, but the point cloud may contain also some points from the ground and understorey.

As shown in the Table 2, *ForestTaxator* possesses a number of advantages as compared to other software. The attractiveness of this software is due to the fact that it provides all necessary functionalities used in the forest inventory field. *ForestTaxator* is freely available to the scientific community and for commercial purposes. We believe the impact of the software on the community will be significant.

9. Conclusions

Today's technical possibilities make it possible to collect data about the environment in the form of three-dimensional scans containing millions of points describing the structure of the scanned objects. There are various methods of performing a scan, including the LiDAR-based methods. Three-dimensional scans are used in many social and scientific fields as well as in civil engineering. In the case of large-area or high-resolution scans, the size of the data may necessitate the use of an appropriate approach to reduce memory

Tab. 2. Comparison between *ForestTaxator* and other free software used for the forest inventory. Standalone means standalone application; DTM – DTM extraction; Stem – tree stem detection; Wood/leaf – point cloud classification into wood/leaf components; Params – Basic tree parameters extraction; QSM – QSM extraction.

Name [reference]	Platform	DTM	Stem	Wood/leaf	Params	QSM
DendroCloud [1]	standalone	+	+	−	+	−
3DForest [35]	standalone	+	+	−	+	+
Computree [34]	standalone	+	+	−	+	+
Simple-Forest [33]	standalone	+	+	−	+	+
AdTree [36]	standalone	−	−	−	+	+
SSSC [37]	standalone	+	+	+	−	−
TreeQSM [11]	Matlab	−	−	−	+	+
TreeLS [38]	R	−	+	−	+	−
TLSeparation [25]	Python	−	−	+	−	−
ForestTaxator	C#	+	+	+	+	+

usage or speed up the search of the set. A scanner using LiDAR is characterized by very high precision, short scanning time and a long range of tens or even hundreds of meters.

Genetic algorithms are a type of heuristic that works well where no other methods are known. They use terminology borrowed from biology, and more specifically from the field of species evolution. The advantage of these algorithms is their great versatility. One basic scheme is always used, which can be extended with additional elements. Another advantage of genetic algorithms is their susceptibility to parallelization. The most illustrative case is the use of multiple populations that may exchange from time to time with a small number of individuals to increase genetic diversity.

We described in detail an attempt to use a 3D point cloud and genetic algorithms to detect and approximate the diameter of the cross-sections of trees. The work covers all stages of data analysis – from loading them into memory to creating the tree model. The work proved that the use of hybrid genetic algorithms for automatic inventory of trees in the forest makes sense – the obtained results were consistent with the reference data to a large extent, while the time of genetic calculations remained very short.

References

[1] J. Elseberg, D. Borrmann, and A. Nuchter. Efficient processing of large 3D point clouds. In *Proc. XXIII Int. Symp. Information, Communication and Automation Technologies ICAT 2011*, pages 1–7, Sarajevo, Bosnia Herzegovina, 27-29 Oct 2011. doi:10.1109/icat.2011.6102102.

[2] G. Krok, B. Kraszewski, and K. Stereńczak. Application of terrestrial laser scanning in forest inventory – an overview of selected issues. *Forest Research Papers*, 81(4):175–194, 2020. doi:10.2478/frp-2020-0021.

[3] A. Bienert, S. Scheller, E. Keane, et al. Tree detection and diameter estimations by analysis of forest terrestrial laserscanner point clouds. In *Proc. Workshop Laser Scanning and SilviLaser 2007 ISPRS 2007*, pages 50–55, Espoo, Finland, 12-14 Sep 2007. `https://www.isprs.org/proceedings/xxxvi/3-w52/final_papers/Bienert_2007.pdf`.

[4] A. Konieczny, and B. Neroj. Projekt działania programu do obliczania miąższości drzew na podstawie danych skanowania naziemnego (TLS). Presentation from "Narada Koordynatorów SIP", Zakopane, 23–25 Feb 2016. `https://www.geomatyka.lasy.gov.pl/documents/25999395/0/Konieczny-TLS.pdf/b13219cc-1608-4004-8692-2de4d0d44a5e`. [Online; accessed 12 Nov 2020].

[5] M. Małaszek. ForestTaxator library. `https://github.com/maciej-malaszek/forest-taxator`. [Online; accessed 10 Dec 2020].

[6] X. Liang, V. Kankare J. Hyyppä, et al. Terrestrial laser scanning in forest inventories. *ISPRS Journal of Photogrammetry and Remote Sensing*, 115:63–77, 2016. doi:10.1016/j.isprsjprs.2016.01.006.

[7] S. Bauwens, H. Bartholomeus, K. Calders, and P. Lejeune. Forest inventory with terrestrial LiDAR: A comparison of static and hand-held mobile laser scanning. *Forests*, 7(12):127, 2016. doi:10.3390/f7060127.

[8] P. Wezyk, K. Koziol, M. Glista, and M. Pierzchalski. Terrestrial laser scanning versus traditional forest inventory first results from the Polish forests. In *Proc. Workshop Laser Scanning and SilviLaser 2007 ISPRS 2007*, pages 12–14, Espoo, Finland, 12-14 Sep 2007. `http://foto.hut.fi/ls2007/posters/Wezyk_ls2007_poster.pdf`

[9] M. Zasada, K. Stereńczak, W. M. Dudek, and A. Rybski. Horizon visibility and accuracy of stocking determination on circular sample plots using automated remote measurement techniques. *Forest Ecology and Management*, 302:171–177, 2013. doi:10.1016/j.foreco.2013.03.041.

[10] R. Astrup, M. J. Ducey, A. Granhus, et al. Approaches for estimating stand-level volume using terrestrial laser scanning in a single-scan mode. *Canadian Journal of Forest Research*, 44(6):666–676, 2014. doi:10.1139/cjfr-2013-0535.

[11] P. Raumonen, M. Kaasalainen, M. Åkerblom, et al. Fast automatic precision tree models from terrestrial laser scanner data. *Remote Sensing*, 5(2):491–520, 2013. doi:10.3390/rs5020491.

[12] P. Wilkes, A. Lau, M. Disney, et al. Data acquisition considerations for Terrestrial Laser Scanning of forest plots. *Remote Sensing of Environment*, 196:153–520, 2017. doi:10.1016/j.rse.2017.04.030.

[13] A. M. Kim, R. C. Olsen, and M. Béland. Simulated full-waveform lidar compared to Riegl VZ-400 terrestrial laser scans. *Laser Radar Technology and Applications XXI*,9832:242–255, 2016. doi:10.1117/12.2223929.

[14] M. T. Vaaja, J. P. Virtanen, M. Kurkela, et al. The effect of wind on tree steam parameter estimation using terrestrial laser scanning. *International Society for Photogrammetry and Remote Sensing Workshop on Laser Scanning*, III-8:117–122, 2016. doi:10.5194/isprsannals-iii-8-117-2016.

[15] D. Seidel, S. Fleck, and C. Leuschner. Analyzing forest canopies with ground-based laser scanning: A comparison with hemispherical photography. *Agricultural and Forest Meteorology*, 154:1–8, 2012. doi:10.1016/j.agrformet.2011.10.006.

[16] M. Dassot, T. Constant, and M. Fournier. The use of terrestrial LiDAR technology in forest science: Application fields, benefits and challenges. *Annals of Forest Science*, 68(5):959–974, 2011. doi:10.1007/s13595-011-0102-2.

[17] L. J. Chmielewski, M. Bator, M. Zasada, et al. Fuzzy Hough transform-based methods for extraction and measurements of single trees in large-volume 3d terrestrial LIDAR data. *In Computer Vision and Graphics: Proc. ICCVG 2010*, Part I, volume 6374 of *Lecture Notes in Computer Science*, pages 265—274, Warsaw, Poland, 20-22 Sep 2010. Springer. doi:10.1007/978-3-642-15910-7_30.

[18] E. Lindberg, J. Holmgren, K. Olofsson, and H. Olsson. Estimation of stem attributes using a combination of terrestrial and airborne laser scanning. *European Journal of Forest Research*, 131(6):1917–1931, 2012. doi:10.1007/s10342-012-0642-5.

[19] W. Zhang, P. Wan, T. Wang, et al. A novel approach for the detection of standing tree stems from plot-level Terrestrial Laser Scanning data. *Remote Sensing*, 11(2):211, 2019. doi:10.3390/rs11020211.

[20] A. T. Albrecht, M. Fortin, U. Kohnle, and F. Ningre. Coupling a tree growth model with storm damage modeling–conceptual approach and results of scenario simulations. *Environmental Modelling & Software*, 69:63–76, 2015. doi:10.1016/j.envsoft.2015.03.004.

[21] D. E. Goldberg. *Genetic Algorithms.* Pearson Education India, 2013.

[22] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996.

[23] A. Burt, M. Disney, and K. Calders. Extracting individual trees from lidar point clouds using *treeseg*. *Methods in Ecology and Evolution*, 10(3):438–445, 2018. doi:10.1111/2041-210x.13121.

[24] K. Olofsson, and J. Holmgren. Single tree stem profile detection using Terrestrial Laser Scanner data, flatness saliency features and curvature properties. *Forests*, 7(12):207, 2016. doi:10.3390/f7090207.

[25] M. B. Vicari, M. Disney, P. Wilkes, et al. Leaf and wood classification framework for terrestrial LiDAR point clouds. *Methods in Ecology and Evolution*, 10(5):680–690, 2019. doi:10.1111/2041-210x.13144.

[26] H. Yoon, H. Song, and K. Park. A phase-shift laser scanner based on a time-counting method for high linearity performance. *Review of Scientific Instruments*, 82(7):075108, 2011. doi:10.1063/1.3600456.

[27] J. D. Schaffer, R. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In Schaffer J. D. (ed.), *Proc. 3rd Int. Conf. Genetic Algorithms*, pages 51–60, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1989.

[28] K. Stereńczak, S. Miścicki, M. Zasada et al. Project *Remote sensing based assessment of woody biomass and carbon storage in forests (REMBIOFOR)*. Funded by the Polish National Centre for Research and Development (NCBiR) within the program Natural Environment, Agriculture and Forestry BIO-STRATEG 2015, under the agreement no. BIOSTRATEG1/267755/4/NCBR/2015. `http://rembiofor.pl/en/305-2/`. [Online; accessed 10 Dec 2020].

[29] M. Małaszek, A. Zembrzuski, and K. Gajowniczek. Supplementary Materials to this paper – file `DetailedResults.xlsx`. Published together with this paper. doi:10.22630/MGV.2022.31.1.2.

[30] M. Małaszek. GeneticToolkit library. `https://github.com/maciej-malaszek/GeneticToolkit`. [Online; accessed 10 Dec 2020].

[31] CloudCompare. 3D point cloud and mesh processing software. Open Source Project. `https://www.danielgm.net/cc/`. [Online; accessed 10 Feb 2021].

[32] T. de Conto, Jean-Romain, C. Hamamura and A. Marcozzi. TreeLS. `https://github.com/tiagodc/TreeLS`. [Online; accessed 10 Feb 2021].

[33] J. Hackenberg, H. Spiecker, K. Calders, et al. SimpleTree—An efficient open source tool to build tree models from TLS clouds. *Forests*, 6(11):4245–4294, 2015. doi:10.3390/f6114245.

[34] J. Trochta, M. Krůček, T. Vrška, and K. Král. 3D Forest: An application for descriptions of three-dimensional forest structures using terrestrial LiDAR. *PLOS ONE*, 12(5):e0176871, 2017. doi:10.1371/journal.pone.0176871.

[35] A. Othmani, L. F. C. Lew Yan Voon, C. Stolz, and A. Piboule. Single tree species classification from Terrestrial Laser Scanning data for forest inventory. *Pattern Recognition Letters*, 34(16):2144–2150, 2017. doi:10.1016/j.patrec.2013.08.004.

[36] S. Du, R. Lindenbergh, H. Ledoux, et al. AdTree: Accurate, detailed, and automatic modelling of laser-scanned trees. *Remote Sensing*, 11(18):2074, 2019. doi:10.3390/rs11182074.

[37] D. Wang. Unsupervised semantic and instance segmentation of forest point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 165:86–97, 2020. doi:10.1016/j.isprsjprs.2020.04.0.

[38] T. De Conto, K. Olofsson, E. B. Görgens, et al. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. *Computers and Electronics in Agriculture*, 143:165–176, 2017. doi:10.1016/j.compag.2017.10.019.