# A GENETIC ALGORITHM FOR CONSTRUCTING HIGH ACCURACY DECISION TREES

A Thesis

by

Elif Ersoy

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Industrial Engineering

Özyeğin University
August 2020

# A GENETIC ALGORITHM FOR CONSTRUCTING HIGH ACCURACY DECISION TREES

Approved by:


Assistant Professor Erinç Albey,
Advisor
Department of Industrial Engineering
*Özyeğin University*

Associate Professor Mehmet Güray Güler
Department of Industrial Engineering
*Yıldız Technical University*

Assistant Professor Enis Kayış,
Co-Advisor
Department of Industrial Engineering
*Özyeğin University*

Assistant Professor M. Yasin Ulukuş
Department of Industrial Engineering
*İstanbul Technical University*


Date Approved: 26 August 2020


Assistant Professor Mehmet Önal
Department of Industrial Engineering
*Özyeğin University*

*To my family for their endless support*

*and*

*to my friends who I consider to be my family...*

# ABSTRACT

Decision trees are one of the most widely used classification methods because of their ease of implementation and explainable nature. Conventional decision tree algorithms construct trees by using myopic, greedy top-down induction strategies. CART (classification and regression tree), ID4, C4.5 are well-known examples of such algorithms. Yet, there are disadvantages of these greedy, myopic methods. One major disadvantage is that while determining next split, they do not consider the possible impact of this decision on future splits. In other words, their myopic nature impedes attaining global optimal solution. In the literature, mathematical programming (i.e., mixed integer programming (MIP) models) is also employed to find optimal trees. Solving constructed models using solvers that guarantee global optimal solution is possible yet remains intractable for even medium size instances due to NP-Hard nature of the problem. This study seeks to construct high accuracy trees than that of the conventional tree construction algorithms, such as CART; and to attain a near-optimal solutions in shorter time than MIP models. To achieve these, we propose a genetic algorithm with a genuine chromosome structure. We also address the selection of the initial population by considering a blend of randomly generated solutions, solutions from the CART, and solutions from the mathematical model, which are constructed for reduced problem instances. We test the performance of the proposed genetic algorithm using five different datasets, with varying bounds on the depth of the resulting trees and using different initial population blends within the mentioned varieties. Results reveal that the performance of the proposed genetic algorithm is superior to that of CART in almost all datasets used in the analysis.

# ÖZETÇE

Karar ağaçları, kolay uygulanması ve yorumlanabilir olması nedeniyle en çok tercih edilen sınıflandırma yöntemlerinden biridir. Geleneksel karar ağacı algoritmaları miyopik ve üstten aşağı doğru indüksiyon stratejisi kullanarak ağaçları inşa eder. CART (sınıflandırma ve regresyon ağacı), ID4, C4.5 bu algoritmalar arasındaki en iyi bilinen örneklerdendir. Ancak, bu yöntemlerin bazı dezavantajları vardır. En önemli dezavantajı, ağaçta sıradaki dallanma kararlaştırılırken gelecekteki kararlara etkileri göz önünde bulundurulmamasıdır. Diğer bir deyişle, bu algoritmaların miyopic doğası optimal çözümün bulunmasını engeller. Literatürde, optimal karar ağaçları oluşturmak için matematiksel modeller oluşturulmuştur (karma-tamsayılı matematiksel modeller). Oluşturulan modellerin optimal sonucu bulması garanti edilir, ancak yapısı gereğince NP-Zor bir problem olmasından dolayı orta büyüklükteki veri setlerinde makul bir sürede optimale ulaşmak zordur. Bu tez, geleneksel ağaçlardan, örneğin CART metodundan daha doğru bir ağaç bulmayı ve karma tamsayılı matematik modelinden daha kısa bir sürede optimale yakın sonucu bulmayı amaçlar. Bu amaçla, doğruluk derecesi yüksek karar ağaçları bulmak için kromozom yapıya sahip bir genetik algoritma öneriyoruz. Ayrıca, başlangıç popülasyonunda, rastgele oluşturulan ağaçların, daha az örnek ile oluşturulan CART algoritmalarının çözümlerinin ve matematiksel modelin çözümlerinin bir karışımına değineceğiz. Önerilen genetik algoritmanın performansı, ortaya çıkan ağaçların derinliğinde çeşitli sınırlar ve belirtilen içeriklerde farklı popülasyon karışımları ile beş farklı veri seti kullanılarak test edildi. Sonuçlar, önerilen genetik algoritmanın performansının analizde kullanılan neredeyse tüm veri setlerinde CART'tan daha üstün olduğunu ortaya çıkarıyor.

# ACKNOWLEDGEMENTS

I would like to extend my deepest appreciation to my supervisors Professor Enis Kayış and Professor Erinç Albey for their patience, consistent support and guidance throughout my journey. Their guidance helped me in all the time of research and study. I also would like to offer my special thanks to all of the professors in the Department of Industrial Engineering for their valuable contributions to my academic background.

I would like to thank my fellow colleagues for all the fun and good times we had during the busy working days. Also, thank my best friends for their help, support and continuous encouragement during my study.

Last but definitely not the least, I owe my deepest gratitude to my family for always believing in me and for their continuous, unconditional love and support throughout my years of study and my whole life. Without their support, I do not think that I could overcome the difficulties.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

Data science has been widely used and has a remarkable role in today's world. Classification is a primary and important approach in data science to understand and interpret unknown data. It is a technique that identifies the categories/labels of unknown observations/data points, and models are built up with the help of a training dataset whose categories/labels are known. There are many types of classification techniques and we can categorize them into two types as linear models and non-linear models. Logistic regression and support vector machines are examples for linear models, while naive bayes classifier, nearest neighbor, decision trees, random forest, and neural networks are nonlinear examples. Also, many of these methods are generated with a greedy approach. Greedy algorithms always decide on the best option at each step, but ultimately that may not guarantee to generate the optimal or less number of rules for a particular dataset.

Among all classification techniques, decision trees (DT) are one of the most commonly used techniques [1],[2],[3]. It is a supervised learning technique that is guided by the training data. DTs recursively partition the training data's feature space through splits and assign a label (class) to each partition. Then the resulting tree is applied to classify unknown future points according to decided splits and labels. However, conventional decision tree methods create each split in each node with greedy approaches and top-down induction methods. This means the possible impact of future splits is not considered while determining each split in the tree. Thus, the underlying characteristic of the entire dataset may not be well captured, and the resulting tree may not be the optimal solution.

Greedy DT construction methods are fast and tractable, however, due to the inadequacies, attempts to formulate optimal [4] and near-optimal [5] decision trees have been discussed for a long time. To solve classification problems, especially for DTs, integer programming is used, however, integer models for classification are NP-complete problems and they are intractable. Optimal decision tree problems resolve the effects of greedy methods by creating the entire decision tree at once to achieve global optimality and in [4] they work on this case. They proposed the Mixed Integer Optimization (MIO) model and it is named as Optimal Classification Tree (OCT). The mentioned model find solutions very slowly or cannot find any solution in a proper time for large and complex datasets in the solver.

Heuristic algorithms can be implemented to find better solutions than greedy approaches and find near-optimal solutions in a reasonable time. Heuristics are adjustable techniques to make quick decisions, especially with complex data and problems. They solve problems by using shortcuts to produce good enough solutions given a limited period of time or deadline. Its name is derived from the Greek word meaning "to discover". So, to deal with the complexity of optimal methods with large instances we can get help from heuristics. Generally, they are applied to solve NP-complete problems and come up with a solution individually or used to provide a good baseline. So, they can build decision trees from scratch or they can increase the performance of constructed trees. Hill-climbing, simulated annealing, tabu search, and genetic algorithms are four of the most popular heuristic algorithms [6].

Heuristic algorithms are applied for optimal decision trees as well. In [3] disadvantages of greedy tree induction are discussed as "Typically, greedy methods are constructed one decision at a time starting at the root. However, locally good but globally poor choices of the decisions at each node can results excessively large trees that do not reflect the underlying structure of the data" and they employ a tabu search algorithm for global tree optimization.

Especially, the genetic algorithm (GA) has been proposed to create decision trees and has been discussed in two different ways to find near-optimal trees. One of them is hybrid or pre-processing methods to select features to be used to construct brand new decision trees [7] and other applies algorithm directly to constructed decision trees to enhance their performances [8]. Therefore, we choose the GA to build up high accurate trees than the greedy approaches due to its compatibility with the DT structure. We will improve local optimal solutions gathered from other tree induction algorithms and with GA we will expand the search area so as not to get stuck in local optima and get close to the near-optimal tree.

In GA applications, different approaches are adopted to form an initial population. In preceding studies, as an initial population randomly generated populations [8],[9] or more intelligent populations [10],[11] are utilized. In this work, we develop a GA to construct high accuracy decision trees for large datasets with a mixture of random and intelligent population. Our intelligent population includes greedy algorithm CART solutions and optimal MIP model OCT [4] sub-solutions for some small instances of the whole dataset. We use a mixture of random and intelligent trees in the initial population to widen search space, improve their performance and reach high accurate trees for large datasets.

In GA, to implement evolutionary operations, it must represent decision trees as chromosome structures. In this work, we propose a linear structure to encode a decision tree to a chromosome and offer an application to generate the initial population. We divide the original dataset into small size subsets then we generate trees with these small subsets on the CART and the OCT model. In GA full-size training data and initial population is used to create more accurate trees. So, we want to use the power of CART and OCT trees in our algorithm and we are trying to improve the performance of provided trees for wider data.

The remainder of the thesis is organized as follows: Chapter 2 provides a brief

review of the DT and GA literature. In Chapter 3, you can find a more detail explanation of our implementation of GA, chromosome structure, and initial population generations. Sections of Chapter 3 present the encoding/decoding decision trees to/from chromosomes, initial population generation, genetic operators like mutation and crossover, and fitness function. In Chapter 4, we will discuss some implementations and corrections over the OCT model, which is proposed in [4]. Chapter 5 explains the experimental details and results. Finally, Chapter 6 concludes the thesis and providing directions for future research.

# CHAPTER II

# LITERATURE REVIEW

In this work, we concentrate on the most widely used classification technique, decision trees. Because of the weaknesses of greedy and optimal tree induction methods, we introduce a heuristic and we develop Genetic Algorithm. We need to express some details and review literature about (1) conventional DTs, (2) optimal DTs, and (3) heuristics applied in DT induction.

## *2.1 Conventional Decision Tree Techniques*

History of DT dates back to the 60s. The first regression tree algorithm in the literature was published by Morgan and Sonquist in 1963, entitled with Automatic Interaction Detection (AID). In AID depending on impurity measures, the algorithm recursively splits data and terminates if reach a certain level of impurity [12]. Then in 1972, Messenger and Mandell, present classification tree algorithm THeta Automatic Interaction Detection (THAID) as an extended version of AID. Although, THAID maximize total number of cases in modal category with recursively splits data. So, AID specialize on regression while THAID specializes on the classification problem [13]. Also, in 1980, Kass proposed Chi-square Automatic Interaction Detection (CHAID) that firstly splitting data into nodes and then integrate nodes that have non-significant splits. The main difference between AID and CHAID is, AID uses a continuous F distribution while CHAID uses the chi-square distribution, appropriate for categorical information [14]. In 1984, Breiman et al. published a more conceived algorithm, Classification and Regression Tree (CART) which follows the same greedy search approach as AID and THAID however, it adds several novel improvements over them. They apply post-pruning over the constructed trees and this

leads to some improvements over accuracy [1]. Thereafter, in 1986 by Quinlan, ID3 is published which uses information entropy to quantify impurity [2], and in 1993 he developed C4.5 as an extended version of ID3 that can control multiple-class decisions [15].

All these mentioned DT methods are based on a greedy approach and suffer from some serious weaknesses besides their useful advantages. Some surveys and studies exhaustively review advantages and disadvantages of conventional DT induction methods. Rokach and Maimon conduct a survey about decision trees that are constructed in top-down induction. They list up some advantages as:

- Decision trees are self-explanatory and when compacted they are also easy to follow. Furthermore decision trees can be converted to a set of rules. Thus, this representation is considered as comprehensible.
- Decision trees are capable to handle both nominal and numeric input attributes.
- Decision trees are capable of handling datasets that may have errors.
- Decision trees are capable of handling datasets that may have missing values.
- Decision trees are considered to be a nonparametric method. This means that decision trees have no assumptions about the space distribution and on the classifier structure. [16]

However, the most important disadvantage is, in top-down induction methods, each split in the tree is determined only for that step, regardless of the potential effects of that decision on future splits. This can lead to trees that cannot capture well key features and characteristics of the dataset, and therefore potentially poor performance when classifying future points [4]. Thus, different approaches are developed to deal with this drawback.

## 2.2 Optimal Decision Tree Induction Techniques

DTs are frequently used in operations research (OR), as a decision analysis tool, to help identify a strategy most likely to reach a goal, also nowadays they are the most popular tool in machine learning [17]. Optimization is a very crucial process in operations research and there are some studies about optimal decision trees. However, in an analysis of optimal decision trees, Hyafil and Rivest [18] report that optimal binary decision tree problem is NP-Complete and Hancock et al. [19] show finding a minimal decision tree consistent with the training set is NP-Hard. Also, finding the minimal equivalent decision tree for a given decision tree [20] and building the optimal decision tree from decision tables [21] are other NP-Hard problems in DT.

Optimal decision tree construction is discussed extensively in the literature with different approaches. Some studies discuss near-optimal trees while others discussed globally optimal. Different approaches can be used to get optimal decision trees, for example, support vector machine (SVM) also produces optimal trees [22]. For global optimality, continuous optimization methods had a significant impact in statistics, however, integer programming had a low effect in statistical computation and mixed integer optimization (MIO) problems are intractable. Nevertheless, classification problems have been generally formulated as an integer and mixed integer optimization problem because there has been an incredible increase in the computational power of MIO solvers over time [4].

Bertsimas and Shioda [23] formulate an integer optimization model named CRIO "to bring to the attention of the statistics and data mining community that integer optimization can have a significant impact on statistical computation". Although CRIO is not able to solve the classification problems to provable optimality for moderately sized classification problems. Bertsimas and Dunn [4] present a very different MIO approach that forms the entire decision tree at a single step to achieve global optimality. Also, Verwer and Zhang [24] propose MIP model that "can be used to learn

good trees up to depth 5 from datasets of size up to 1000". Günlük et al. [25] present MIP formulation to achieve good accuracy with small trees using moderately-sized training sets.

Even though the power of solvers increases day by day, large datasets and deeper trees remain a problem for optimization models. As Bertsimas and Dunn mention in their study [4], solving integer models with wider depth or larger dataset size the rate of finding solutions is slower, and also considerable time is required. Therefore, because of the time cost, in studies [4], [24] and [25] solving time is limited for MIP model and sub-optimal solutions for a given instance are obtained. Thus, proposed models cannot prove global optimality and cannot find optimal in a given time limit.

## *2.3 Use of Heuristics in Decision Trees*

Phyu [26] notes: "The problem of constructing optimal binary decision trees is an NP-complete problem, and thus theoreticians have searched for efficient heuristics for constructing near-optimal decision trees". So, the use of heuristic with decision trees is discussed widely in the literature and four of the most popular heuristic search algorithms are hill-climbing, simulated annealing, tabu search, and genetic algorithms [6]. For continuous feature data, evolutionary design is suggested in [27], Genetic Algorithm is proposed in [28] and an extreme point tabu search algorithm is proposed in [3]. Also, [3] is an example of heuristics in optimal decision trees and Gehrke et al. [29] develop a bootstrapped optimistic algorithm for decision tree construction.

GAs take natural evolution as an example on its operations [30],[31],[32]. They are one of the most widely used heuristics for DTs [7],[28],[8],[10],[33],[9]. They are not only for constructing new trees, but it also used in different approaches, such as for selecting features to be used to construct a new decision [7],[34],[35], for selecting optimal training data to be used [36], and for improving the performances of

constructed trees [8],[37].

Additionally, for initializing GA, there are some different approaches for initial population generation. Generally randomly generated initial populations are used [8], [9] and rarely a more intelligent populations are used with the help of greedy algorithms [10], [11].

## *2.4 Our Contributions*

In this work, we proposed the Genetic Algorithm to enhance the performances of constructed trees. We construct initial trees with the CART and OCT model, unlike the current studies. We want to improve accuracies of CART trees and find near optimal trees with the help of OCT with less time cost. Also, we want the proposed approach to be convenient with small and large datasets, thus our approach is suitable for all sizes and provide solution in reasonable time. The present research explores, for the first time, the effects of optimization approach while initializing the GA. So, we work with CART [1] and OCT [4] jointly and CART separately in the initial population to detect benefits of adding OCT to the initial population.

# CHAPTER III

# PROBLEM STATEMENT & METHODOLOGY

In this thesis study, we propose an algorithm to classify datasets in the form of $(\mathbf{X}, \mathbf{Y})$. Each set containing $n$ data points $(\mathbf{x_i}, \mathbf{y_i})$, $i = 1, \ldots, n$. Also, each data point includes $p$ features and $K$ possible labels is assigned to this point in total. $\mathbf{x_i}$ is the feature vector, and $\mathbf{x_i} \in [0, 1]^p$ and $y_i$ is class labels, $y_i \in \{1, \ldots, K\}$.

To get accurate trees in proper time, we focus on the Genetic Algorithm. We take greedy trees from CART [1], also feasible trees created with the optimal decision tree induction model, OCT [4]. We want to work on greedy trees because they decide each split decision, which is best only for the current step, and then it will categorize whole data by that decision. But this split may not be the best and informative split for the entire data, so that with these split decisions we may achieve local optima only, not globally optimal. So, with the help of GA, we expect to get over these local optima and search through the other solutions and find near-optima. Also, we use Bertsimas and Dunn's MIP mathematical model, OCT [4], to get some feasible trees. This OCT model results in an optimal tree, but it takes a long time to achieve optimality, also especially for the complex and large datasets it takes much more time to get and prove an optimal solution. So, we use this model only to get intermediate feasible tree solutions for subsets to improve them with GA. We use subsets to reduce instance size, thus with reduced size, we will get better qualified results in a shorter time. Though for small size datasets, their subsets might result in an optimal tree and for large datasets, subsets cannot give optimal results but might result in a better solution than the whole size case in the same time limit. Moreover, we made some modifications and corrections over this OCT model. We will explain details about

CART and OCT in the following subsections.

## *3.1 Genetic Algorithm*

A genetic algorithm is inspired by theory of natural evolution and genetics. It is an adaptive heuristic search algorithm and widely used to generate high-quality solutions for optimization problems. In our GA implementation, we concentrate on the following facts: (1) new chromosome structure, (2) initial population, (3) fitness function, (4) selection algorithm, (5-6) genetic operations, (7) generation and (8) stopping condition and overall flow of GA. These are explained in detail in the following subsections to understand the essentials of our GA.

### 3.1.1 Encoding the Solution Representation (Chromosome Structure)

In GA, chromosome representation is essential to describe each individual and critical to make relevant searches in solution space. Tree-based representation for the candidate solution is the best one and natural because of the structure of binary split trees [9]. To decode each chromosome as a feasible tree, the structure must contain all split decisions and rules in a consistent order.

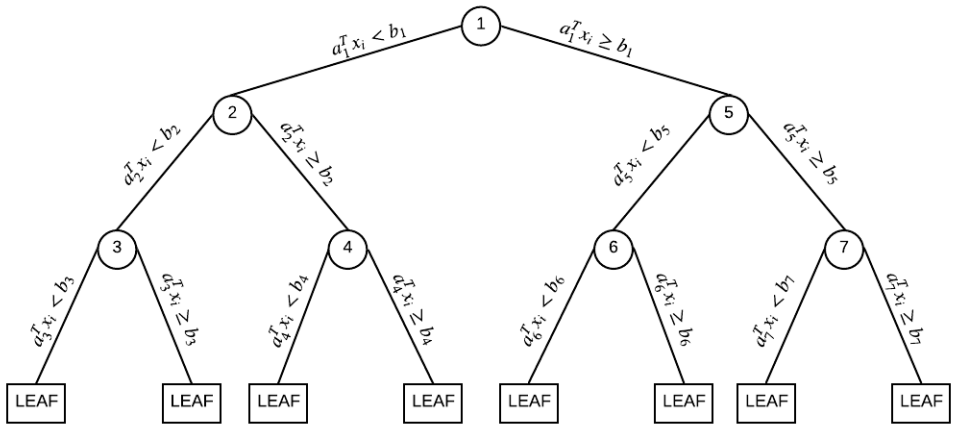


**Figure 1:** Tree structure and split rules at depth 3.

The system of the decision tree is, each branch node (also known as an internal node) represents a rule on a feature (attribute), each branch stand for an outcome of the rule, and each leaf node (also known as a terminal node) holds a label of class. So, each branch node stores a split rule, and data points follow the related branch based on that rule. Also, in this work, we concentrated on binary split trees so there are only two branches in each node splits, which means splits are bidirectional. The left branch refers less than operation and the right branch refers greater than or equal to operation (see Figure 1) and each decision node stores continuous threshold value for each feature. So, in GA's solution representation, if we encode which feature is used as a rule in each branch node, and its threshold values, we will construct a tree with this linear representation. Then with these split rule notations we will decode each tree and calculate the fitness value easily. To encode that representation, we use a modification of Jankowski and Jackowski [9] as follows.

| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ |
|---|---|---|---|---|---|---|
| $b_1$ | $b_2$ | $b_3$ | $b_4$ | $b_5$ | $b_6$ | $b_7$ |

**Figure 2:** Chromosome structure for tree in Figure 1.

Unlike to structure in [9], in our study, each branch node is sorted in a depth-first manner as an implicit data structure in two arrays (see Figure 2). All feature IDs ($a_t$) are maps to an integer and each feature threshold values ($b_t$) are maps to normalized continuous values between $[0, 1]$, so each component is numeric for each branch node $i$. In the chromosome, we do not want to store class labels for each leaf node. We assign labels to the leaf nodes at the end of the training procedure, with the class that occurs most often among training data points in the leaf node according to split rules.

Chromosome dimension is $2 \times (2^D - 1)$, where $D$ is the maximum depth of the tree, so there are two rows with length $2^D - 1$. The first-row of these two rows denotes feature information for each split and second-row represents threshold values

for the related feature. It has "$2^D - 1$" number of columns which is equal to the number of total branch nodes. In our algorithm, we define $D$ as a fixed parameter, while chromosome length is also fixed. Also, with the help of restrictions in crossover operation (see subsection 3.1.5), length cannot change after evolutionary operators as well. Thanks to the fixed depth and fixed chromosome length, we will restrict uncontrolled growth on trees. Also, to be more specific, initial trees do not have to be grown by given maximum depth ($D$). They will be in lower depths, so we assign 0 (zero) to branch nodes if there is no split here.

In this study we use the strict structure for each tree, thus the structure of a defined tree is: the right branch always follows the "$\geq$" rule and the left branch follows "$<$" rule. Thus, it is enough to store feature and threshold value for each node, to construct each tree easily. Also, CART trees and OCT trees follow similar rule structure, they all recursively partition feature space with bidirectional splits and right branch for "$\geq$" rule and left branch for "$<$" rule. So, this chromosome representation fits our initial population trees completely, and that is why we work with these trees.

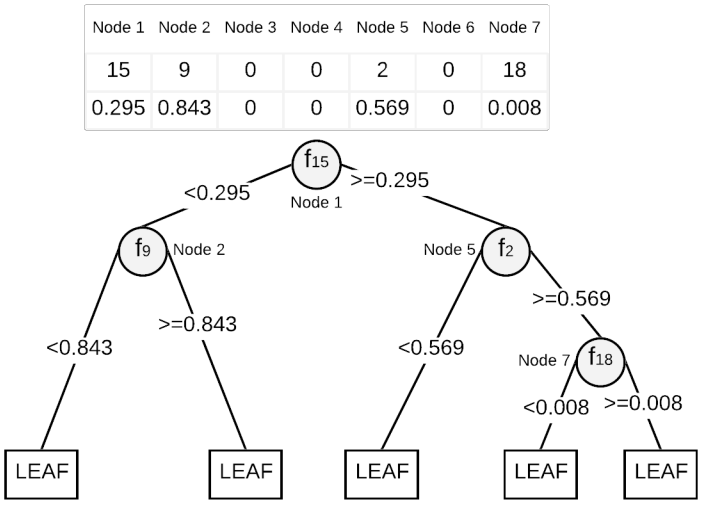


**Figure 3:** Tree structure and split rules at depth 3.

Regardless of the tree type, even if it is coming from CART or OCT, all trees are in the same format. One of the decision tree at depth 3 is represented in Figure 3. To explain it in detail, the split feature at the root node (node 1) is the $15^{th}$ feature and the threshold is 0.295. So, in training data, data points whose $15^{th}$ feature is less than 0.295, follow left branch, accumulated in node 2 and points greater than or equal to 0.295, follow right branch and accumulated in node 5. The second column of the chromosome represents the rule of the left node, node 2, then split feature is $9^{th}$ feature and the threshold is equal to 0.843. Thus, data points accumulated in node 2 follows that rule, they split according to their $9^{th}$ feature and if the value is less than 0.843 they follow the left branch and accumulated in node 3. Points greater than or equal to 0.843 follow the right branch and accumulated in node 4. But, in node 3 and node 4, there is not any splitting, thus related indices in the chromosome take a value 0. Then points collected in node 3 and node 4 are accumulated in the right leaf nodes because $0 \geq 0$ is satisfied at right branches and left branch rule is violated, $0 < 0$.

### 3.1.2 Generation of Initial Population

Initial population/solutions for GA are generated by CART [1] and OCT [4]. In GA initial population needs solution space with solutions more than 1, so as Fu et al. [10] use in their paper, we create some sub-trees for the initial population. In this work, we generate sub-trees using three ways: randomly, with the CART algorithm and with the OCT mathematical model.

For random trees, we assign a random rule for each node. So, we randomly choose features and threshold values. The threshold is a continuous value within the interval $[0, 1]$. We use values between 0 to 1 because we normalized each integer feature for all datasets, thus threshold must be between these values.

For sub-CART trees, a whole size training data is divided into some number

of subsets randomly with a decided instance size (explained in Section 5.1.3), and CARTs are generated for these subsets. Some data points can be in multiple subsets, so these subsets are not distinct. When CART trees are constructed for these subsets, we call them as sub-trees or sub-CART. Later original data will be classified using these sub-trees and fitness value for each sub-tree is calculated for the whole dataset.

We use sub-OCT trees as well, and we generate them with the same procedure as sub-CART trees. But there is a small difference: for sub-CART, we take a final solution from the CART algorithm for subsets and it takes only seconds, but for sub-OCT trees, we cannot reach optimal trees even for subsets of large datasets in an acceptable time. So, we limit run time and get intermediate feasible solutions from the model. We split the whole training data into subsets because with large datasets solver cannot find any feasible solution up to 2 hours. So, with subsets, we reduce the instance size and reach sub-optimal feasible trees for large size datasets. We will discuss details about the OCT model in **Chapter 4.**

As an initial population, we try 3 different mixtures. These mixtures include (1) random trees only, (2) random trees plus sub-CARTs, and 1 full-CART solutions and (3) random trees plus sub-CARTs plus sub-OCTs, and 1 full CART. Details of these population types and effects of these selections are discussed in **Chapter 5.**

### 3.1.3 Fitness Value

Fitness function is a particular type of objective function for GA. Our study is a part of a classification problem; thus, our objective is finding accurate trees, in other words, we want to maximize classification accuracy or minimize misclassification error. So, in this work fitness value will be calculated as the total number of misclassified points or total correctly classified points of each individual tree. In classification trees, leaf nodes are labeled with the majority points' class labels after split rules are determined. Therefore, in each leaf node minority points are labeled incorrectly, and

the total number of these minority points is equal to the misclassification error and others are labeled correctly.

In our algorithm, we prefer to maximize classification accuracy as an objective. Thus, our fitness value is equal to the total number of correctly classified points, which is similar to accuracy. We choose this one because of the working principle of our selection algorithm, roulette wheel selection. We explain it in the following section in more detail.

When a new offspring is reproduced after evolutionary operations, fitness value needs to be recalculated. Firstly, points are assigned to related leaf nodes according to new split rules. Then, each leaf is labeled with majority class label, so points labeled incorrectly according to their real class labels, are called misclassified points and others labeled as correctly classified points. Then new fitness is calculated with the total number of correctly classified points.

In the GA algorithm, we calculate the fitness function value of each individual sub-tree with the whole training data instead of subsets, because, in this paper, the classification tree is generated for the whole dataset which considers all data points completely.

### 3.1.4 Parent Selection

To come up with a new generation, the algorithm selects some parents from the current generation based on the fitness value and then produce children (offspring). In GA and other evolutionary algorithms, well-known selection methods are tournament selection, roulette wheel selection, rank-based selection, etc. These methods can select each individual in the population more than once, so each individual possibly spread its genes to more than one child.

In our GA implementation, we use the roulette wheel (RW) selection. Roulette wheel selection is under the Fitness Proportionate Selection Methods family. In each

selection method of this family, every solution has a probability of being selected according to their fitness values proportion. Especially, in the roulette wheel, fitness values of each tree in the current generation take a place in the wheel according to their weighted fitness value. Thus, high fitness value takes a wider proportion and it will be selected with a higher probability. Furthermore, we want to choose trees with high accuracy as a parent to spread its genes to the next generations. This will help us to reproduce stronger children. So, if fitness value is used as a total number of correctly classified points, the roulette wheel works well and will choose high accurate trees with higher probability. Also, the roulette wheel selection method cannot be used on minimization problems, that is why our objective is maximization and fitness is total correctly labeled points. So, to select stronger parents, which has fewer misclassed points and more correctly classified ones, the roulette wheel method is the proper one. To deal with a convergence problem in roulette wheel selection, we apply some restrictions. We forbade the selection of the same parent in each run, also mutation and different crossover options are providing diversity on children.

### 3.1.5 Crossover Operation

Crossover operator breeds children by combining pieces of selected parents in the current population. The operation begins with the selection of two individuals (parents) with the RW method. A random cut point is selected on each parent according to a randomly generated number. Possible cut points are between 1 (root node) to n (total number of tree nodes) (see Figure 4 and Figure 5). After identifying the random cut points in both parents, new individuals (offspring) are created by replacing sub-part form the first parent by the one from the second parent. We cut each parent from the exact same point/node and swap the same sized cut-part. Figure 6 illustrates an example of the crossover operation.

Crossover is applied in two ways, 1-point, and 2-point crossover. In a 1-point

crossover, we choose one random cut point from the parent's chromosomes. The cut-part starts from the cut point and runs through the last node/end of the chromosome, and then these sub-parts are exchanged. Whereas in 2-points crossover, which is like one-point but instead of one, we select two random cut points. So, selected cut-part is the interior sub-part between these selected cut points, then these interior parts swapped with each other. In the proposed algorithm, we cut sub-trees at the same level from each parent, in other words, we exchange exactly the same sub-parts from each parent. Also, in a 1-point and 2-point crossover, selected sub-parts are always starting from any intermediate branch node to the terminated node. Where this intermediate node has to be the ancestor of the terminated node.

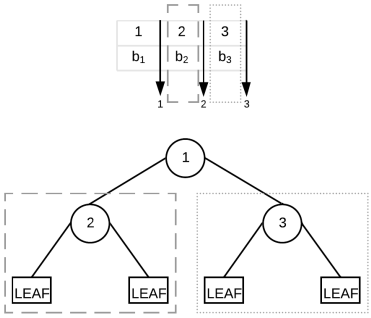


**Figure 4:** Possible crossover cut parts at depth 2; 1-2 or 2.

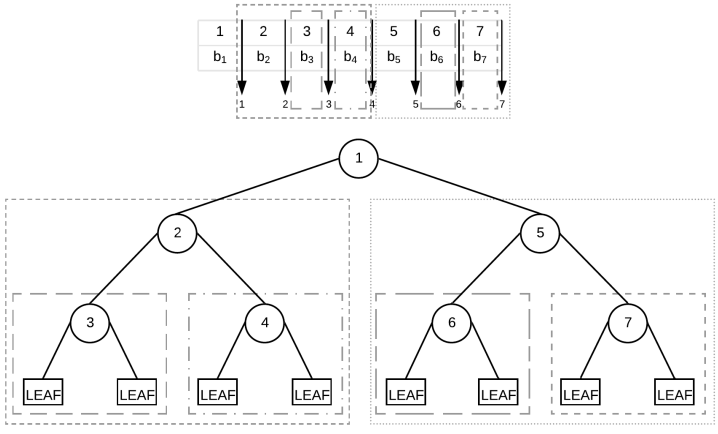


**Figure 5:** Possible crossover cut parts at depth 3; 1-4 or 2-3 or 3-4 or 4 or 5-6 or 6.

In Figure 4, possible cut points for depth 2 trees are shown. 1-point crossover is applied when parents are cut from the $2^{nd}$ cut point and crossed each other. 2-point crossover is applied when the part between the $1^{st}$ and $2^{nd}$ cut point is crossed. To be more precise, after the crossover operation, the maximum depth of the tree cannot change because we cut each parent from the exact same point. Figure 5 shows cut points for depth 3 trees. For 1-point crossover at depth 3, possible cut points will be 4 or 6 and for 2-point crossover, potential parts will be between 1-4, 2-3, 3-4 or 5-6. These specific cut parts represent each meaningful sub-tree in a maximal tree. This specification also prevents depth change in the crossover.

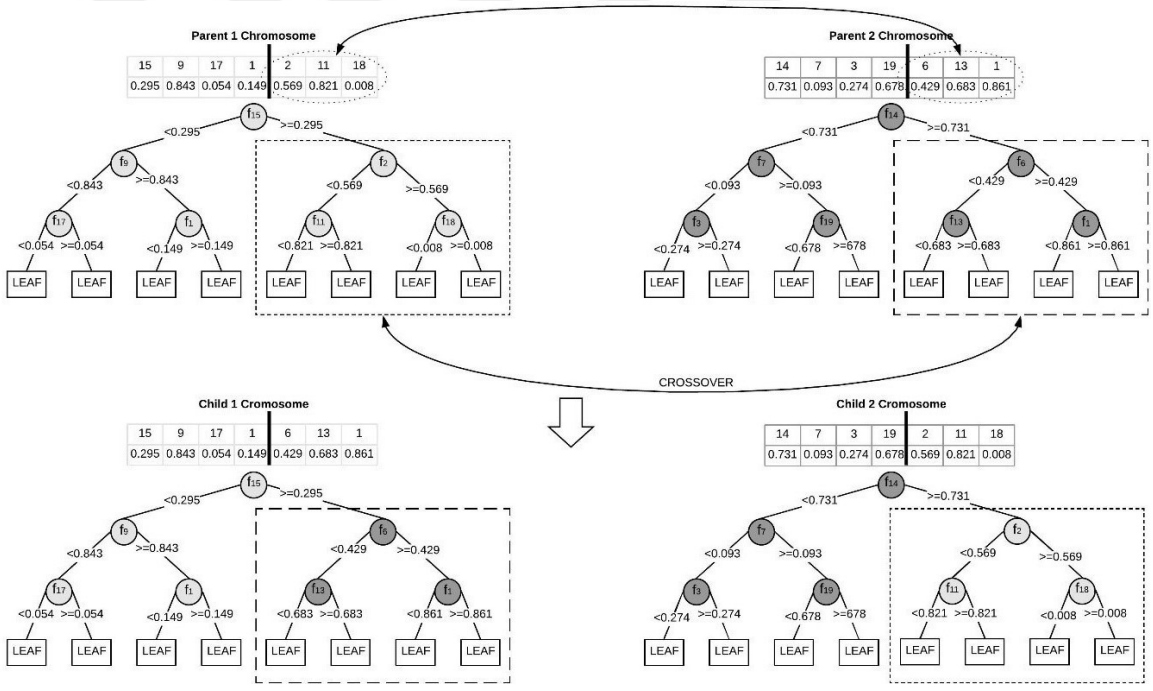


**Figure 6:** Crossover operation on selected parent and generated children at depth 3 trees.

In Figure 6, you can see the illustration of 1-point crossover on the depth 3 tree. We cut each parent from the $4^{th}$ cut point, which is decided randomly in the algorithm, and cut parts swapped between each other. This operation gives birth to newly generated 2 children, leaf labels of these children are also updated. The fitness

function is calculated for these children with these new labels.

### 3.1.6 Mutation Operation

Mutation operation is applied after new children birth. It creates a change over individuals in the population. In the proposed algorithm, the mutation is applied based on a defined rate (mutation rate), on to the randomly selected node of the generated child. With the help of mutation, GA enables searching broader space with providing genetic diversity [9]. In the proposed implementation, we use node base mutation similar to Jankowski and Jackowski [9], we randomly change both feature and split threshold of a randomly selected node (see Figure 7).

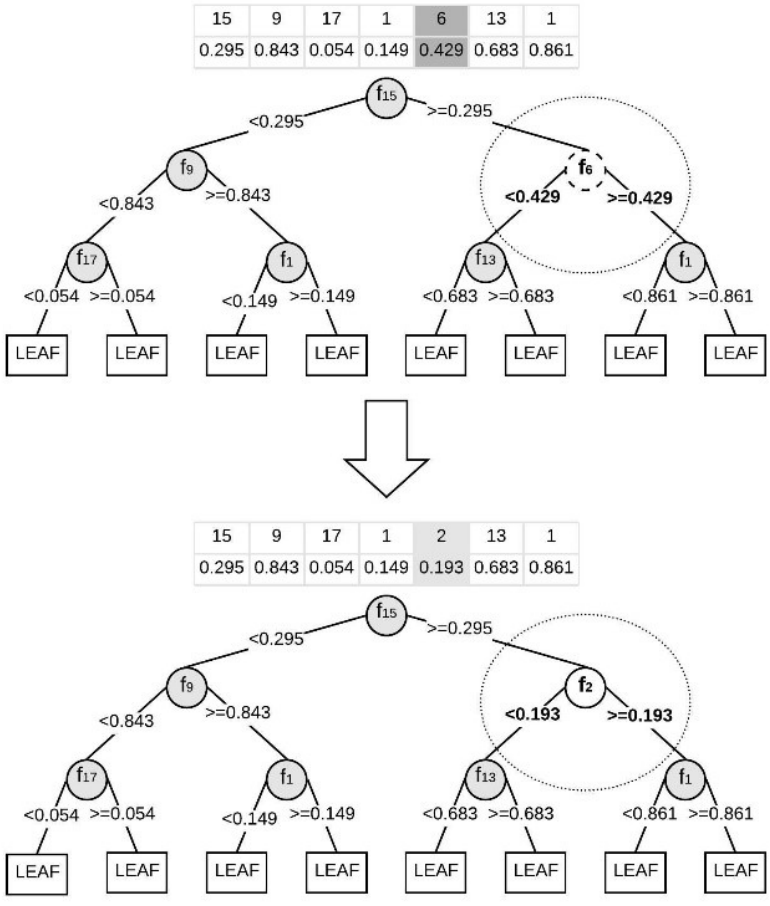


**Figure 7:** Tree node mutation.

Mutation operation is illustrated in Figure 7, it shows how mutation might change Child 1, which comes from the crossover we define in Figure 6. In this illustration, the $5^{th}$ node is randomly selected to mutate, and this node's feature ID and the threshold value are changed with random values. With mutation, the class assignment of leaf nodes may also change and updated based on the majority class on each node because the split rule is changed. Also, there is a restriction, the mutation will be applied to all branch nodes except the root node (1st node).

### 3.1.7 Replacement and New Generation

In general GA applications, to generate the new population, all the individuals of a current population are replaced with the new individuals which are generated using reproduction over the current population. But, it may cause loss of the best individuals because of its stochastic nature. In the proposed GA, we applied the elitism procedure to create new generations. Elitism keeps a proportion (known as the *eliteRate*) of the fittest individuals into the next generation with no revision. For example, given a population of 100 individuals, if you have an elite rate of 10%, you transfer the top ten best individuals of the current generation to the next generation with no modification. Then, the rest of the generation is filled up with the children generated by crossover and mutation. So, the rest of the 90 individuals are offspring coming from the 45 crossover operations, since at each crossover operation, 2 children are born. And then we apply mutation to some portion (mutation rate) of these children under the mutation rate.

### 3.1.8 Stopping Criteria and Overall Algorithm

There are quite different applications of the GA in the literature. The following flowchart in Figure 8 shows the general steps of our GA implementation.

To explain our algorithm in more detail, the steps of GA implementation is as follows:

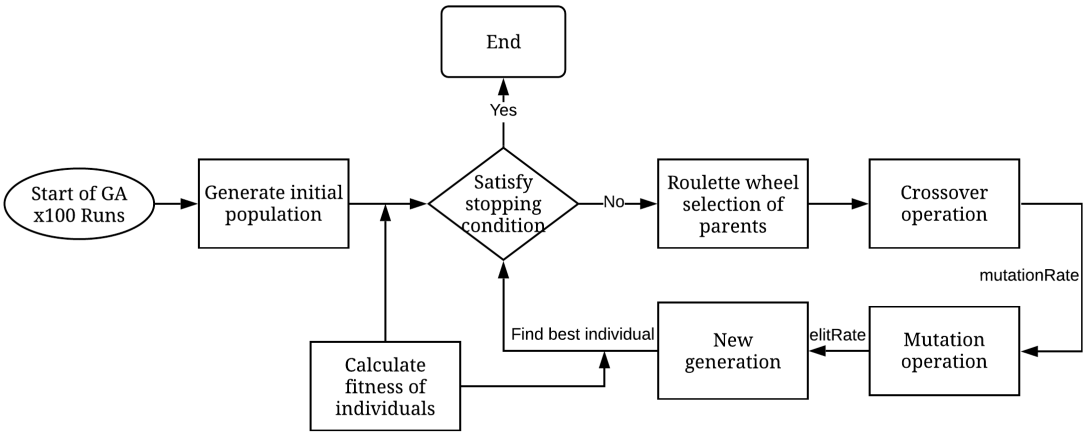


**Figure 8:** Flow of GA.

Repeat below steps TotalRun times:

1. Generate an initial population (first generation) of size $N$ with the selected mixture.

2. Initialize a variable, *Iter*, for keeping track of successive iterations where the best tree found in each generation has not changed.

3. Repeat while $Iter < MaxIter$:

   (a) Select $eliteRate*N$ best individuals from the previous generation and keep them into the new generation.

   (b) Repeat until the new generation is fulfilled to size $N$.

      i. Randomly select two members from the previous generation based on the roulette wheel method.

      ii. Run the crossover operator: input selected two members and obtain two children.

      iii. Add new children to the new generation

iv. Run the mutation operator: input one of the currently generated children from crossover and apply mutation if *mutationRate* is satisfied.

v. Replace mutated child in the new population.

(c) If the new generation cannot provide a better tree and fitness value than the previous generation's best, increase *Iter* by 1.

4. Select the best tree in the final generation and its fitness.

We repeat these steps $TotalRun$ times to minimize the effects of randomization on overall performance. Also, we choose the final solution as the best tree found in all $TotalRun$ runs.

Algorithm 1 shows steps of proposed algorithm in pseudo code format. Parameters used in this algorithm, $MaxIter$, $elitRate$, $mutationRate$, $TotalRun$, are explained in more detail at Chapter V, and exact values are presented.

**Algorithm 1** Genetic Algorithm for Decision Tree Induction

generate an initial population of size $N$ : $inPop$
initialize $Iter = 0$
initialize generation $Gen = inPop$
**for** $i = 1, \ldots, TotalRun$ **do**
    **while** $Iter < MaxIter$ **do**
        compute fitness of each individual in $Gen$
        number of elitism: $ne = N * elitRate$
        keep best $ne$ trees in $Gen$ and save them in $Gen_E$
        number of crossover $nc = (N - ne)/2$
        **for** $c = 1, \ldots, nc$ **do**
            select two parent tree solutions from $Gen$ using a roulette wheel, $T_{P1}$ and $T_{P2}$
            apply crossover operator on $T_{P1}$ and $T_{P2}$ to generate children $T_{C1}$ and $T_{C2}$
            select $T_M$ from $T_{C1}$ or $T_{C2}$ then apply mutation operator, over one of the node of $T_M$ under the rate $mutationRate$ and generate $T'_M$
            update selected child $T_M$ with $T'_M$
            save $T_{C1}$ and $T_{C2}$ to $Gen_C$
            update $c = c + 1$
        **end for**
        update $Gen = Gen_E + Gen_C$
        select best individual with maximum fitness and call it $T^*$
        **if** fitness of $T^*$ > fitness of $T_{best}$ **then**
            update $T_{best} = T^*$
            $Iter = 0$
        **else if** fitness of $T^* \leq$ fitness of $T_{best}$ **then**
            $Iter = Iter + 1$
        **end if**
    **end while**
**end for**
**return** the best solution $T^*$ over $TotalRun$ replication

# CHAPTER IV

# EXTENSION: CORRECTIONS AND MODIFICATIONS ON OPTIMAL CLASSIFICATION TREE (OCT) MODEL

Well known decision tree induction algorithms can only generate local optimal solutions because of their greedy, top-down nature. In [4], Bertsimas and Dunn generate a mathematical model named Optimal Classification Tree (OCT) as the Mix Integer Optimization problem to solve the problem that CART seeks to solve as a formal optimization problem. They state that, some number of discrete decisions are required to make while constructing trees. The most natural representation is made with following decisions in their paper:

- At every new node, we must choose to either branch or stop.
- After choosing to stop branching at a node, we must choose a label to assign to this new leaf node.
- After choosing to branch, we must choose which of the variable to branch on.
- When classifying the training points according to the tree under construction, we must choose to which leaf node a point will be assigned such that the structure of the tree is respected. [4]

Using MIO to formulate optimal decision tree problem helps to model all of these decisions in a single problem. Thus, this allows decision events that are not considered in isolation as opposed to greedy methods. In fact, it allows the tree to be constructed by considering the full effect of the decision in the first step. Also, this avoids the need for pruning and impurity measures [4].

In classification problems especially in supervised learning, the bias-variance trade-off is a fundamental problem. This problem is about balance between the underfitting and overfitting. With classification problems, ideally, it is desirable to create a model that accurately captures characteristics of training data and generalizes future data well. In OCT model, multi-objective function is used which consider tradeoff between accuracy and complexity of the tree. This trade-off is controlled by the some parameters and also they restrict minimum number of points assign to each leaf node. Complexity of the tree is considered as total number of branch nodes.

In their study, firstly, to generate an optimal decision tree, maximum depth, $D$ is determined and a maximal tree is created. Depth is used to define the number of edges from the leaf node to the tree's root node and at depth $D$, tree has $T = 2^{(D+1)} - 1$ nodes where they index by $t = 1, \ldots, T$.

In proposed model, notation $p(t)$ is used to represent the parent node of node $t$, and $A(t)$ to denote the set of ancestors of node $t$. Also, $A(t)$ is consist of two sets as $A_L(t)$ and $A_R(t)$, such that $A(t) = A_L(t) \cup A_R(t)$. $A_L(t)$ is the set of ancestors of node $t$, whose left branch has been followed on the path from the root node to node $t$, and $A_R(t)$ is the set of right-branch ancestors. Also, nodes are divided into two sets as, $T_B$ and $T_L$, where $T_B$ is branch for nodes and $T_L$ is for leaf nodes. Branch nodes apply a split and leaf nodes make a class prediction for each point that falls into that leaf [4].

Based on the maximum depth and maximal tree for this depth, branch and leaf node sets are given to the optimization model. Then the model determines split rules, feature and threshold value, for each branch node. In their proposed model, they restrict model to univariate decision trees, thus each split at each node should only involve a single variable. Also, as other optimal decision tree related papers used, this paper also use an objective function that is trying to minimize misclassification error besides minimizing the number of splits.

Following sets, indices, parameters and decision variables are used to set proposed model. You can see details of the related optimization model below.

Sets:

$T_L$ : leaf nodes of the maximal tree

$T_B$ : branch nodes of the maximal tree

$A_L(t)$ : left ancestors of node $t$

$A_R(t)$ : right ancestors of node $t$

$p(t)$ : parent nodes of node t

Indices:

$i$ : data point

$j$ : feature

$k$ : label

$t = m$ : node

Parameters:

$Y_{ik}$ : 1 if point $i$ is labeled as $k$, -1 otherwise

$x_{ij}$ : $j^{th}$ feature of point $i$

$N_{min}$ : minimum number of points required in any leaf node

$\epsilon_{max}$ : maximum error value

$\epsilon_j$ : error value for $j^{th}$ feature

$\alpha$ : complexity parameter

$\widehat{L}$ : coefficient to normalize the misclassification against the baseline accuracy

Decision Variables:

$L_t$ : number of misclassified points in node $t$

$N_t$ : total number of points in node $t$

$N_{kt}$ : total number of points labeled as $k$ in node $t$

$c_{kt}$ : 1 if leaf node $t$ labeled as $k$, 0 otherwise

$d_t$ : 1 if split applied at node $t$

$z_{it}$ : 1 if point $i$ is in node $t$

$l_t$ : 1 if node $t$ contains any point

$a_t$ : track the split at node $t$, it is a vector where $a_{jt}$ take a value 1 if $j^{th}$ feature is used in node $t$ as a split rule

$b_t$ : track the split at node $t$, define threshold value for node $t$

The mathematical model of OCT is on the next page and explained in detail on the following page.

$$\min \quad \frac{1}{\widehat{L}} \sum_{t \in T_L} L_t + \alpha \sum_{t \in T_B} d_t \tag{1a}$$

$$\text{s.t.} \quad L_t \geq N_t - N_{kt} - n(1 - c_{kt}) \qquad k = 1, ..., K, \ \forall t \in T_L \tag{1b}$$

$$L_t \leq N_t - N_{kt} + n c_{kt} \qquad k = 1, ..., K, \ \forall t \in T_L \tag{1c}$$

$$L_t \geq 0 \qquad \forall t \in T_L \tag{1d}$$

$$N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it} \qquad k = 1, ..., K, \ \forall t \in T_L \tag{1e}$$

$$N_t = \sum_{i=1}^{n} z_{it} \qquad \forall t \in T_L \tag{1f}$$

$$\sum_{k=1}^{K} c_{kt} = l_t \qquad \forall t \in T_L \tag{1g}$$

$$\sum_{j=1}^{p} a_{jm} x_{ij} \geq b_m - (1 - z_{it}) \qquad i = 1, ..., n, \ \forall t \in T_L, \ \forall m \in A_R(t) \tag{1h}$$

$$(1 - d_m)\epsilon_{max} + \sum_{j=1}^{p} a_{jm}(x_{ij} + \epsilon_j) \leq \ b_m + (1 + \epsilon_{max})(1 - z_{it})$$

$$i = 1, ..., n, \ \forall t \in T_L, \ \forall m \in A_L(t) \tag{1i}$$

$$\sum_{t \in T_L} z_{it} = 1 \qquad i = 1, ..., n \tag{1j}$$

$$z_{it} \leq l_t \qquad t \in T_L \ i = 1, ..., n \tag{1k}$$

$$\sum_{i=1}^{n} z_{it} \geq N_{min} l_t \qquad t \in T_L \tag{1l}$$

$$\sum_{j=1}^{p} a_{it} = d_t \qquad t \in T_B \tag{1m}$$

$$0 \leq b_t \leq d_t \qquad t \in T_B \tag{1n}$$

$$d_t \leq d_{p(t)} \qquad t \in T_B / \{1\} \tag{1o}$$

$$z_{it}, l_t, c_{kt} \in \{0, 1\} \qquad i = 1, ..., n, \ k = 1, ..., K, \ \forall t \in T_L \tag{1p}$$

$$a_{jt}, d_t \in \{0, 1\} \qquad j = 1, ..., p, \ \forall t \in T_B \tag{1q}$$

This mathematical model constructs a tree for given instance by determining the following decisions: Is there any split at node $t$ ($d_t$); and then if any split applied at $t$, what is the related feature with that split and what the threshold value is ($a_{jt}, b_t$); while minimizing the total misclassification error ($\sum L_t$) and total split applied ($\sum d_t$) in the tree. Other decision variables are calculated with the help of $d_t, a_{jt}, b_t$, which are the main constituents for tree construction.

Split applied at node $t \in T_B$ is tracked with decision variables $a_t \in \mathbb{R}^p$ and $b_t \in \mathbb{R}$. Elements of $\mathbf{a}_t$ is setting to be binary variables in this model and setting summation of elements equal to the 1 to enforce univariate splits in the tree. Constraint (1m) enforce this restriction by using binary variable $d_t$ which tracks is there any split at branch node $t$. If there is a split at $t$ then $d_t = 1$ thus this constraint ensures that $\mathbf{a}_t$ has one element that is 1, otherwise, $d_t = 0$ which means no split is applied on branch node $t$ and they model this by setting $\mathbf{a}_t = 0$ and $b_t = 0$. Also, constraint (1n) and (1q) support these decisions. As we apply in our algorithm, in this study, they also normalized feature vector such as $\mathbf{x}_i \in [0, 1]^p$, thus constraint (1n) define interval and restrict values of $b_t$, that is threshold value for the split rule at node $t$.

Decision trees have a hierarchical structure that does not allow for splitting at that node if there is no split in its parent node. Constraint (1o) ensures that the created model fits this structure and prevents a node from splitting if its parent is not split.

Allocation of data points to each leaf node is tracked with binary variable $z_{it}$. This indicator is equal to 1 if $\mathbf{x}_i$ is in leaf node $t$ and 0 otherwise. Also, binary variable $l_t$ indicates if leaf node $t$ contains any point, and it is equal to 1 if there is any, 0 otherwise. Constraint (1k) linked up these two indicators, and make $l_t = 1$ if there are one or more points in leaf node $t$. Constraint (1l) uses $l_t$ and enforces a minimum number of points to assign at each leaf with the $N_{min}$ value. Thus, $N_min$ is the minimum number of points at leaf node if this leaf is open. With constraint

(1j), each point is forced to be assigned to one and only one leaf.

In proposed tree construction model, strict structure is used. Left branch is always for $<$ rule and right branch is for $\geq$ rule. Thus, when assigning points to leaf nodes, constraint (1h) and (1i) enforcing points follows the splits that are required by this structure. In proposed model constraint (1h) is for left split and (1i) is for right split. However, strict inequality is not supported by MIO, thus some conversions are applied to this constraint and change it to non-strict inequality, such as constraint (1h). They add $\epsilon$, which is small constant, to the left-hand side of constraint (1h). They specify a different $\epsilon_j$ for each feature $j$ to make $\epsilon$ as big as possible without affecting feasibility. They decide largest valid value for each $\epsilon_j$ by sorting the values of the $j^{th}$ feature and take smallest non-zero distance between sequential values. Which $\epsilon_j$ to use is chosen according to the feature used in the split at that node and guided by the $a_m$. Also, coefficient of $(1 - z_{it})$ in each constraint is specified as 1 in constraint (1h) and as $(1 - \epsilon_{max})$ in constraint (1i), which are the largest possible values.

They define the matrix $\mathbf{Y}$ which is provided as a parameter, and $Y_{ik} = 1$ if label of point $i$ is $k$, such that $y_i = k$, and $Y_{ik} = -1$ otherwise. Main objective of this problem is minimizing the misclassification. Thus incorrect label prediction must be penalized with 1. To calculate the total number of misclassified points in each leaf node, the number of points at each, as well as the number of points of label $k$ must be known. With constraint (1e) and (1f), they set $N_{kt}$ to the total number of points of label $k$ in node $t$ and also $N_t$ to the total number of points in node $t$.

Each leaf node can be labeled with only one label, where constraint (1g) ensures this restriction is satisfied. This assignment is based on the most common class labels among all points assigned to the leaf node. In proposed model, to track label assignment of each leaf, they define a binary variable $c_{kt}$, and it is equal to 1 if label of leaf $t$ is $k$. Number of misclassified points, or in other words, optimal misclassification loss in each node is denoted with $L_t$, which is equal to the number of points in the

node minus the number of the most common label ($L_t = N_t - \max_k\{N_{kt}\}$). Constraint (1b),(1c) and (1d) is used to linearize calculation of $L_t$, because in MIO strict equality cannot be used.

As we mentioned at the beginning of this chapter, bias-variance trade-off is fundamental concept in classification. Thus in the objective (1a) of this model, they made regulation with complexity of the tree. Their objective is multi-objective function, where the first component is misclassification cost ($\sum_{t\in T_L} L_t$) and second component is for the complexity of the tree which is defined with the total number of splits ($\sum_{t\in T_B} d_t$) and there is trade-off between each other. They balance and tune each component with some coefficients such as $\widehat{L}$ and $\alpha$. $\widehat{L}$ is equal to the total number of misclassification, if predicting the most common class for the entire tree. And also they tune $\alpha$ with some tuning algorithm, but we made some modification over it and modified algorithm is described later.

Up to this point, we define general concept and working principle of OCT model as it described in [4]. However, when we examine this mathematical model as it presented in [4], we noticed some typos, errors and inconsistencies. These typos are about indice denotations and defined sets. Also, there is some inconsistencies about denotations and set declaration between model part and in-text definitions. Above model is corrected version and we apply the following corrections:

Node indices, $t$, is defined over an inaccurate set in constraint (1k) and (1l).

$$\text{wrong}: \quad z_{it} \leq l_t \qquad t \in T_B \qquad (1k')$$

$$\text{corrected}: \quad z_{it} \leq l_t \qquad t \in T_L,\ i = 1, ..., n \qquad (1k)$$

$$\text{wrong}: \quad \sum_{i=1}^{n} z_{it} \geq N_{min} l_t \qquad t \in T_B \qquad (1l')$$

$$\text{corrected}: \quad \sum_{i=1}^{n} z_{it} \geq N_{min} l_t \qquad t \in T_L \qquad (1l)$$

$l_t$ and $z_{it}$ are decision variables, which are defined for tracking assigned points to each leaf node. So, constraint (1k) and (1l) cannot be defined over the branch node set ($T_B$). In their paper, they made typos in the constraint definition part. Also, constraint (1k) must be defined for all $i = 1, ..., n$ and this definition is missing in the model.

Defined set for t is miswritten at constraint (1h) and (1i) as well.

$$\text{wrong}: \quad \sum_{j=1}^{p} a_{jm} x_{ij} \geq b_t - (1 - z_{it}) \qquad i = 1, \ldots, n,\ \forall t \in T_B,\ \forall m \in A_R(t) \quad (1\text{h}')$$

$$\text{corrected}: \quad \sum_{j=1}^{p} a_{jm} x_{ij} \geq b_m - (1 - z_{it}) \quad i = 1, \ldots, n,\ \forall t \in T_L,\ \forall m \in A_R(t) \quad (1\text{h})$$

$$\text{wrong}: \quad \sum_{j=1}^{p} a_{jm}(x_{ij} + \epsilon_j) \leq b_t + (1 + \epsilon_{max})(1 - z_{it})$$

$$i = 1, \ldots, n,\ \forall t \in T_B,\ \forall m \in A_L(t) \quad (1\text{i}.1')$$

$$\text{corrected}: \quad \sum_{j=1}^{p} a_{jm}(x_{ij} + \epsilon_j) \leq b_m + (1 + \epsilon_{max})(1 - z_{it})$$

$$i = 1, \ldots, n,\ \forall t \in T_L,\ \forall m \in A_L(t) \quad (1\text{i}.1)$$

In these constraints, $t$ define each leaf node and $m$ define branch nodes that are left or right ancestors of leaf $t$. Thus, $t$ must be defined over $T_L$ and $b_t$ must be denoted as $b_m$.

Although, constraint (1h) and (1i) are for split decision and tree construction, in other words, these are keystones of tree algorithm. They decide split rule $(\mathbf{a}_m, b_m)$ and target leaf for each point $(z_{it})$ according to rule. But we realize another problem with one of these constraints when we solve this mathematical model in MIO solver. Solver set all decision variables related to split rules $(\mathbf{a}, b, d)$ to 0, which means that

there is not any branch and split in the tree. However, each data point is assigned to whichever leaf nodes they prefer according to their class labels, even if there is not any split. So, this makes misclassification and the total number of splits equal to zero, which means the objective is equal to 0 and solver claims this solution is an optimal and feasible solution for a defined model. But this is not a proper solution in the real case, because points cannot be assigned to any leaf nodes at no split case. In other words, $a_m = 0$ and $b_m = 0$ if a branch node $m$ does not apply a split. As we explain in the previous chapter and as they consider in this model, the proposed tree structure follows a strict rule, where the left branch always follows $a_m x_i < b_m$ and right branch follows $a_m x_i \geq b_m$ rule. So, if $a_m = 0$ and $b_m = 0$ (means there is no split at node $m$), then this has the effect of forcing all points, which are arrived in node $m$, to follow the right branch split. Therefore, the condition for the left split is never satisfied as $0 < 0$ and they must follow right branch [4].

In the proposed model, constraint (1h) is for the left branch and constraint (1i) is for the right branch splits. After some detailed examinations, we noticed that the source of the mentioned problem is constraint (1i.1). Constraint (1i.1) overcome assignment rule and violates $0 < 0$ rule, as $z_{it}$ (equal to 1 if point $i$ is assigned to leaf node t, and 0 otherwise) can take any value whether $a$ and $b$ is zero or larger than 0. This means that point $i$ at node $m$ can be assigned to leaf node $t$ which is succeeding from the left branch of node $m$ nevertheless there is not a split (at the leaf node's ancestor nodes). But this constraint (1i.1) should not allow this assignment to be made because this assignment violates constraint (1i.1) as $0 < 0$. This means, if there is no split at node $m$, points cannot follow left branch. Since the objective is minimization, then solver set all $a$ and $b$ to zero, which also means there is not any tree because there are not any branches. To deal with this problem, we modify constraint (1i.1) as this way:

$$\text{old}: \quad \sum_{j=1}^{p} a_{jm}(x_{ij}+\epsilon_j) \leq b_m + (1+\epsilon_{max})(1-z_{it}) \quad i=1,\ldots,n,\ \forall t \in T_L,\ \forall m \in A_L(t) \tag{1i.1}$$

$$\text{modified}: \quad (1-d_m)\epsilon_{max} + \sum_{j=1}^{p} a_{jm}(x_{ij}+\epsilon_j) \leq b_m + (1+\epsilon_{max})(1-z_{it}) \quad i=1,\ldots,n,\ \forall t \in T_L,\ \forall m \in A_L(t) \tag{1i}$$

We add new component $(1-d_m)M$ to the left-hand side of the constraint (1i). This extension forces LHS greater than 0 if there is any split at branch node $m$ ($\sum a_{jt}, d_m = 1$), then assignment can be made to leaf node $t$ which follows that branch ($z_{it}=1$). However, even if $\sum a_{jt}=0$, which means there is not any split rule and constraint (1m) enforce $d_m$ to 0, then LHS of constraint (1i) is equal to $bigM$ and $z_{it}$ must be equal to 0. This prevents mistaken assignment to left branch leaf in the tree when there is no splitting at the ancestors. In other words, this resolve the violation problem at $0<0$ rule. Also, to specify values for the $bigM$, the largest possible value is $\epsilon_{max}$. Because at the no split case $(d,a,b=0)$, to enforce $z_{it}=0$, LHS cannot be greater than RHS but must be larger than 0.

Also, the objective of this model is a multi-objective function. So, as used in multi-objective functions some coefficients are needed to normalize each part. $\alpha$, which is a complexity parameter, is used as a coefficient of a split decision, and $\widehat{L}$ is used to normalize the misclassification against the baseline accuracy. $\widehat{L}$ is equal to misclassification when predicting the most popular class for the entire dataset. The CART solution is used to find proper $\alpha$ (complexity parameter), and some validation steps are applied. In this paper, to tune $\alpha$ we do not use the same search algorithm as in paper [4]. We made some modifications over the proposed method, to reduce the run time of tuning step. We use below modified steps for $\alpha$ tuning:

1. Set the maximal depth for the MIO problem, $D_{max}$, and the minimum leaf size $N_{min}$.

2. For D = 1, . . . , $D_{max}$:

   (a) Run CART with train data, using $N_{min} = n \times 0.05$ with $\alpha = 0$, *maxdepth* = D. Collect all complexity parameters related to split numbers for maximal depth (we call it as the set of CP).

   (b) For C = CP:

      i. Run CART (*rpart*) with train data again, using $N_{min} = n \times 0.05$ with $\alpha$ = C, *maxdepth* = D

      ii. Add the CART solution to the warm start pool.

   (c) Post-process the solution pool to remove all solutions that are not optimal to mathematical model (OCT) for any value of $\alpha$.

   (d) Identify the best performing solution on a validation set, and the range of $\alpha$ for which this solution is optimal. Use the midpoint of this interval as the tuned value of $\alpha$.

For CART trees we use rpart package in R. After tuning the $\alpha$ , we use the tuned value on the objective function of the OCT model to take feasible solutions. In their paper they also solve OCT with training data besides CART, thus this causes a lot of time wasted.

# CHAPTER V

# EXPERIMENTAL ANALYSIS & COMPUTATIONAL RESULTS

To inspect and show the performance of the proposed study, firstly we explain our experimental design with provide details about datasets, parameters, and initial population, then the computational results of the proposed approach on different variations of datasets are discussed. We compare the performance of the proposed GA to that of CART as a benchmark.

## *5.1 Experimental Design*

### 5.1.1 Datasets

We use five different datasets with different dimensions and characteristics to evaluate the effects of data size, number of features, and class over the GA performance. We obtain these datasets from the UCI machine learning repository [38]. Dimension details about each dataset are described below (see Table 1).

**Table 1:** Details of datasets used in experiments.

| **Dataset Name** | **# Points** *(n)* | **# Features***(p)* | **# Classes***(k)* |
|---|---|---|---|
| Wine | 173 | 13 | 3 |
| Chess | 3196 | 36 | 2 |
| Image Segmentation | 210 | 19 | 7 |
| Parkinson | 756 | 754 | 2 |
| Madelon | 2600 | 500 | 2 |

Datasets are given to GA in the form of **(X, Y)** where **X** is feature vector and **Y** is the label, they contain $n$ observations, each observation is defined with $p$ features and $K$ possible class labels. The $x$ values for each feature across the data are continuous and normalized between 0 and 1, which means each $x_i \in [0, 1]^p$. Unity-based

normalization is applied as is described in the following formulation (8). OCT model [4] is also formulated according to normalized feature values.

$$X'_j = \frac{x_j - x_{j_{min}}}{x_{j_{max}} - x_{j_{min}}} \qquad \forall j = 1, \ldots, p \tag{8}$$

In this study, we do not have any dataset with categorical features, but if desired to use such a dataset, we can apply dummy encoding method. This method is widely used to convert categorical features into continuous form. Every level in each categorical feature are decomposed to dummy features. Then existence of a level is represented by 1 in corresponding dummy feature and absence is represented by 0. However, with this representation number of features are increased by the total number of categories.

We test the effectiveness of our proposed algorithm, with 5-fold cross-validation. Cross-validation ensures that every observation from the whole size dataset has the chance to appear in training and test set, hence the model ends up with less biased results. And then we take an average of 5 folds‘ results. Actually, we use validation data only at $\alpha$ tuning step, but in all other steps, as training data, we use a combination of training and validation data. In other words, at the beginning, we have training, validation, and test sets, but all steps except $\alpha$ tuning, we assume that training data is a summation of training and validation.

### 5.1.2 Parameters

In the proposed GA, there are some parameters that affect the performances of our algorithm. Here are the four most important parameters: (1) $MaxIter$ which is used for restricting the ineffective improvement moves, so this is our stopping condition to terminate our algorithm, (2) $eliteRate$ represents the proportion of best individual to keep in next generation, (3) $mutationRate$ is the proportion for mutation operation applied on children and (4) $TotalRun$ which is the total number of replications.

$MaxIter$ is the most important parameter in the algorithm. It prevents unnecessary operations. The algorithm terminates if there are no better trees for $MaxIter$ consecutive generations. To give more details, the iteration counter increases with each sequential ineffective generation. However, if the best tree in the new generation improves the best solution and updates the best tree so far, then the iteration counter is reset. If the iteration counter reaches $MaxIter$ value, it means there is no better solution for the last successive $MaxIter$ generations, and GA ends.

The proposed algorithm replicated $TotalRun$ times to get final results because there are lots of randomization in reproduction operators as selection, crossover and mutation. So, we apply some replications to decrease the bias effect of randomness over each procedure. Other parameters are described in the previous chapter.

Presented results in section 5.2, were taken with parameter values shown in Table 2. These parameters are tested with detailed performance analysis and below values are selected as the values that yield the best performance.

**Table 2:** Values of each parameter.

| **Parameter** | **Value** |
|---|---|
| *MaxIter* | 10 |
| *elitRate* | 0.2 |
| *mutationRate* | 0.1 |
| *TotalRun* | 100 |

### 5.1.3 Initial Population

In this work, we aim to improve the performance of GA by using different initial population mixtures. To initialize GA, we test three different population blends. These populations are created with trees that are constructed with 3 different approaches: (i) randomly generated trees, (ii) CART based trees, and (iii) OCT-based trees. These CART and OCT trees are generated with subsets of the whole training datasets and only 1 CART tree is constructed with the full-size training set.

When subsets are used in CART and OCT, solutions found for each subset are called as sub-trees, such as sub-CART and sub-OCT. Then whole training data classified using these sub-trees. Leaf labels and fitness value for each sub-tree is calculated for the whole training dataset as well in the GA algorithm.

In each population, the total number of each sub-tree is $m$. In other words, we construct $m$ different subsets from the training data. For datasets with $< 1,000$ data points, each $m$ subsets include 15% of train data, and for the dataset with $\geq 1,000$ data points, each $m$ subsets include 5% of train data. We decrease the rate of subset size when train data size increases. Because solver cannot find any solution for OCT in a given time limit with large data size, so we need fewer datapoints. We determine this $m$ value after some analysis, and we choose $(0.2 \times n)$, where n is the size of the whole data. A brief comparison of different percentages and sizes is discussed in the next section. Also, in Table 3, you can see details about each population mixtures and their contents.

**Table 3:** Population types and their contents.

| **Population** | **Content** |
|---|---|
| GA-R | $m$ Random Trees |
| GA-RC | $m$ Random + $m$ Sub-CART + 1 full-CART Trees |
| GA-RCO | $m$ Random + $m$ Sub-CART + 1 full-CART Trees + m Sub-OCT |

#### *5.1.3.1* ***Random Tree Generation***

We generate each random tree by assigning random rules for a given depth. Random feature ID for each node, which is an integer, is assigned between $[0, p]$ and the threshold value, which is continuous between $[0, 1]$ because we normalized each data feature values to 0-1 interval. Thus, we do not need to check the pattern of each feature value.

#### *5.1.3.2* ***sub-CART Generation***

For constructing CART trees, we used the *rpart* package [39] in the R programming language version 3.5.3 [40] on RStudio Version 1.2.1335. In the *rpart* function, there is a control parameter called *minbucket*, which limits the minimum number of observations in any leaf node to prevent overfitting. We use this parameter as $(0.05 \times n)$ for all datasets. Also, the $N_{min}$ parameter in the OCT model, which is similar to *minbucket*, is equal to $(0.05 \times n)$ as well.

We have already explained how to construct sub-trees but to give more detail, we divide whole training dataset into $m$ subsets randomly with a decided instance size (0.15 or 0.05 of whole data). Some data points can be in multiple subsets, so these subsets are not distinct. Then CARTs are generated for each of these subsets respectively. We get the final tree for each subset from the *rpart*, and also we get one tree that is trained with whole training data. Then we compound our initial population with these more intelligent trees.

#### *5.1.3.3* ***sub-OCT Generation***

For the OCT model, which is MIP and its logic is explained in the previous chapter in detail, we use *Cplex 12.10*. Also, with version 12.10, on Mixed Integer Problems, this solver's performance improved by 10% on average [41]. We tune $\alpha$ for all datasets respectively as described in Chapter 4.

We are using the same subsets created in the sub-CART generation step to construct sub-OCTs. OCT model has some time complexity over large sized datasets. Because of the difficulty of the model, which is $n \cdot 2^D$, at higher depth and large data size, the rate of finding solution is slower, and more time is required [4]. Thus we use reduced instance sizes to decrease this difficulty as much as possible. Also, in our algorithm, we use OCT to get any feasible solution, in other words we are not trying to solve it optimally. For small size datasets, we will get optimal OCT solution for

each or some subsets. However, for large size datasets, even for reduced size subsets we cannot reach optimality in general, thus we get the best feasible tree solution within a time limit.

Also, for large datasets with more features, we apply feature selection/elimination as well to get a better feasible solution in a shorter time. Although the number of data points decreases by dividing the data into some smaller subsets, thus difficulty metric is decreased, the excess feature number also makes it hard to find a feasible solution in a short time. So, for each subset, we also select features based on feature importance. Details of this feature selection are as follows and Algorithm 2 shows general steps of the selection algorithm.

**Algorithm 2** Feature Selection via Variable Importance

classify full training data with CART
calculate variable importance of each feature $j$ by CART and assign each to $Imp_j$,
generate $m$ subsets ▷ same subsets used in sub-CART generation
**for** $j = 1, \ldots, p$ **do** ▷ calculate total importance values
    $totalImp = totalImp + Imp_j$
**end for**
**for** $t = 1, \ldots, m$ **do** ▷ select $f$ feature for each subset $t$
    $sum = totalImp$
    **for** $i = 1, \ldots, f$ **do**
        $rand =$ random number in the range $[0, sum]$
        $partialSum = 0$
        $index = 1$
        **while** $partialSum < rand$ **do**
            $partialSum = partialSum + Imp_{index}$
            $index++$
        **end while**
        **return** $index$
        select related feature $f$ of $Imp_{index}$
        add feature $f$ into $selectedFeatures$ set
        $sum = sum - Imp_{index}$
    **end for**
    eliminate all features except $selectedFeatures$ in the subset $t$
    **return** updated subset $t$
**end for**
**return** all updated subsets ▷ use these subsets at sub-OCT generation

We calculate all features' importance for training data, then for each subset, we select a given number of features with a method like a roulette wheel. So, we use different selected feature mixtures for each subset, thus we get literally different trees. We select features without replacement, so each subset includes the same number of different features. However, in GA we use whole size training and testing data. We reduce row and feature size with subset and feature selection, because OCT cannot find or prove optimal solution up to 2 hours for large datasets as it mentioned in [4]. Thus, splitting data into subsets and reduce feature size provide us a small instance that results in better qualified trees in a reasonable time.

In [4], they state that for large size datasets, they use 2 hours limit to ensure that solver could make progress, but there is no evidence to reach optimality for large datasets within the time limit. Thus, we use 2 hours as a time limit as well. However, we divide 2h (7200 sec) by the number of trees ($m = 0.2 \times n$) and distribute equally to each tree ($7200/m$), then we use this as a time limit for each sub-OCT. For example, if we have $m = 30$ subsets, then we execute OCT model just for 240 sec and then terminate solver and take the best feasible solution. Although, because of the well-defined multi-objective function and trade-off in OCT, these feasible solutions perform well in quality.

We add intelligent trees, or in other words trees with intermediate performance, into the initial population and our aim is to outperform greedy tree solutions such as CART. Because of that aim, we also include a full-CART tree into the populations besides sub-trees, thus we can make an improvement over the benchmark result in GA. We will evaluate the contributions of the CART and OCT-based initial population in the next section.

## *5.2 Experimental Analysis and Result*

In this section, we share and discuss some results and comparisons about our GA with the proposed experimental design. We aim to find more accurate trees rather than the greedy approach, and our benchmark algorithm is CART. Also, we want to use the power of optimization technique OCT in the initialization step of our proposed approach. We use three different population mixtures with different sizes to test the impacts of extensions in the initial population (see Table 3). The proposed GA is codded in Java language (Java version 1.8.0) and computed on Eclipse IDE version 4.14.0. All computations are done in a PC with Intel Core i7-8550U @ 1.80 GHz processor and 8 GB RAM.

Our first initial population includes $0.2 \times n$ different random trees. And second initial population type includes the same $0.2 \times n$ random trees plus $0.2 \times n$ sub-CART and 1 full-sized CART solution. The third initial population type includes additional $0.2 \times n$ sub-OCT solutions besides second population. We use abbreviations for each population as defined in Table 3 (R, RC, RCO). Proposed algorithm is trained with each dataset at 4 different depths as $\{2, 3, 4, 5\}$, and we compare them respectively. These depths are given as a maximum depth limit. For example, if the selected depth is 5, then in the population and reproduction steps some trees will be in the lower depths but maximum depth must be 5.

We choose the amount of each tree type in the initial population as $0.2 \times n$. We also analyzed results with $\{0.05, 0.1, 0.2, 0.3\} \times n$ and also fix size 30. Accuracy performances boost with the increase in the percentages, and fixed-size provide low performance in large datasets. We choose 0.2, because after this percentage, the amount of improvement is decreasing, also 0.2 and 0.3 provide very similar results. But, for large datasets, when this percentage increases population size also increases. Then, a large initial population raises the run time rapidly, so 0.2 is the best one within the tested ratios in terms of performance and run time.

In this section, the proposed results are averages of 100 replications. We replicate GA 100 times to minimize the effects of randomization since there are lots of randomly selected operations in the proposed algorithm. Operations such as parent selection, crossover, mutation are based on random processes. In the result tables, we introduce four different methods. "CART" refers to pure CART solution obtained from *rpart* and trained with the full training set, the other three methods are versions of proposed GA. These methods differ in initial population types. In each table, we share outcomes of "Mean Out-Sample Accuracy", "Mean In-Sample Accuracy Improvement", "Initial Population Generation Time" and "GA Time" for each experiment.

"Mean Out-Sample Accuracy" refers to average accuracies of best trees in every 100 runs, and it is also an average of 5 folds. We use average out-sample accuracies across 5 folds to compare the performance of our algorithm in a cross-validation manner. Also, instead of demonstrating exact in-sample accuracies, we share "average in-sample accuracy improvement". This is the difference between the average in-sample accuracy out of 100 runs and in-sample accuracy of the initial best accurate tree in the initial generation, and again it is an average of 5 folds. Thus, we will show how much our proposed GA improves initial trees. Also, we share total elapsed time in two components "Pop. Gen. Time" and as "GA time". We report time in two parts, because 2 initial population types include intelligent trees, and we spent some significant amount of time to construct these trees. Thus, "Pop. Gen. Time" is for a total elapsed time to construct initial population and "GA Time" is the total time in seconds for 100 replications but an average of 5 folds. Table 4 presents these in-depth direct comparisons at depth 2, Table 5 at depth 3, Table 6 at depth 4 and Table 7 at depth 5.

At all depths, GA always performs better than the pure CART regardless of the population type. In Table 4, results of depth 2, two out of five datasets show the RCO population makes GA stronger than pure CART, and stronger than GA with RC and

**Table 4:** Full results at depth 2.

| DATASET | | | | | | | |
|---|---|---|---|---|---|---|---|
| Name | n | p | K | | Mean Out-Sample Accuracy | Mean In-Sample Accuracy Improvement | Pop. Gen. Time (sec) | GA Time (sec) |
| Wine | 173 | 13 | 3 | CART | 88.78% | | | (0.112) |
| | | | | GA-R | 85.49% | +4.92% | | 0.340 |
| | | | | GA-RC | **89.32%** | +0.61% | 0.302 | 0.734 |
| | | | | GA-RCO | 88.82% | +1.14% | 66.513 | 1.101 |
| Chess | 1396 | 36 | 2 | CART | 76.69% | | | (0.131) |
| | | | | GA-R | 86.05% | +8.37% | | 165.186 |
| | | | | GA-RC | **86.92%** | +0.00% | 6.113 | 171.702 |
| | | | | GA-RCO | **86.92%** | +0.00% | 196.731 | 283.902 |
| Image Segment. | 210 | 19 | 7 | CART | 37.14% | | | (0.169) |
| | | | | GA-R | 55.46% | +6.13% | | 0.4092 |
| | | | | GA-RC | 58.20% | +2.10% | 0.584 | 1.009 |
| | | | | GA-RCO | **59.06%** | +1.60% | 2149.584 | 1.540 |
| Parkinson | 756 | 754 | 2 | CART | 79.89% | | | (0.349) |
| | | | | GA-R | 77.98% | +1.13% | | 5.652 |
| | | | | GA-RC | 81.44% | +0.07% | 15.123 | 12.628 |
| | | | | GA-RCO | **81.58%** | +0.06% | 7215.123 | 19.878 |
| Madelon | 2600 | 500 | 2 | CART | 65.19% | | | (0.783) |
| | | | | GA-R | 58.59% | +3.46% | | 164.193 |
| | | | | GA-RC | **66.33%** | +0.70% | 45.724 | 279.817 |
| | | | | GA-RCO | 66.01% | +0.61% | 5740.956 | 550.582 |

⋆ The best performing solutions for each dataset are highlighted in **bold**.

⋆ Time in parenthesis shows elapse time of full CART in *rpart*.

**Table 5:** Full results at depth 3.

| DATASET | | | | | | | |
|---|---|---|---|---|---|---|---|
| Name | n | p | K | | Mean Out-Sample Accuracy | Mean In-Sample Accuracy Improvement | Pop. Gen. Time (sec) | GA Time (sec) |
| Wine | 173 | 13 | 3 | CART | 91.02% | | | (0.214) |
| | | | | GA-R | 85.29% | +8.30% | | 0.587 |
| | | | | GA-RC | 92.02% | +1.01% | 0.316 | 1.261 |
| | | | | GA-RCO | **92.15%** | +1.43% | 179.712 | 1.974 |
| Chess | 1396 | 36 | 2 | CART | 90.43% | | | (0.249) |
| | | | | GA-R | 92.51% | +13.09% | | 293.306 |
| | | | | GA-RC | **93.55%** | +0.41% | 7.073 | 303.062 |
| | | | | GA-RCO | 93.52% | +0.37% | 3984.079 | 488.044 |
| Image Segment. | 210 | 19 | 7 | CART | 52.38% | | | (0.254) |
| | | | | GA-R | 63.68% | +12.61% | | 0.799 |
| | | | | GA-RC | 75.38% | +3.54% | 0.597 | 1.895 |
| | | | | GA-RCO | **75.64%** | +4.92% | 7200.597 | 3.155 |
| Parkinson | 756 | 754 | 2 | CART | 80.95% | | | (0.526) |
| | | | | GA-R | 79.12% | +1.56% | | 14.290 |
| | | | | GA-RC | 82.64% | +0.77% | 15.472 | 37.391 |
| | | | | GA-RCO | **82.67%** | +0.88% | 7215.472 | 61.690 |
| Madelon | 2600 | 500 | 2 | CART | 69.04% | | | (1.004) |
| | | | | GA-R | 63.23% | +2.76% | | 1441.673 |
| | | | | GA-RC | 69.20% | +0.36% | 55.311 | 618.512 |
| | | | | GA-RCO | **69.53%** | +0.53% | 7255.311 | 7326.187 |

⋆ The best performing solutions for each dataset are highlighted in **bold**.

⋆ Time in parenthesis shows elapse time of full CART in *rpart*.

**Table 6:** Full results at depth 4.

| DATASET | | | | | | | |
|---|---|---|---|---|---|---|---|
| Name | n | p | K | | Mean Out-Sample Accuracy | Mean In-Sample Accuracy Improvement | Pop. Gen. Time (sec) | GA Time (sec) |
| Wine | 173 | 13 | 3 | CART | 91.02% | | | (0.214) |
| | | | | GA-R | 86.17% | +3.78% | | 1.115 |
| | | | | GA-RC | **92.14%** | +1.09% | 0.330 | 2.433 |
| | | | | GA-RCO | 91.88% | +1.51% | 613.847 | 3.314 |
| Chess | 1396 | 36 | 2 | CART | 89.86% | | | (0.391) |
| | | | | GA-R | 93.88% | +12.44% | | 575.263 |
| | | | | GA-RC | 94.08% | +0.01% | 7.686 | 519.844 |
| | | | | GA-RCO | **94.09%** | +0.01% | 7207.686 | 979.006 |
| Image Segment. | 210 | 19 | 7 | CART | 69.05% | | | (0.222) |
| | | | | GA-R | 71.40% | +10.67% | | 1.605 |
| | | | | GA-RC | 77.83% | +1.62% | 0.687 | 2.992 |
| | | | | GA-RCO | **79.20%** | +2.63% | 7200.687 | 5.063 |
| Parkinson | 756 | 754 | 2 | CART | 80.95% | | | (0.715) |
| | | | | GA-R | 78.50% | +1.79% | | 29.580 |
| | | | | GA-RC | 83.19% | +1.49% | 15.669 | 81.482 |
| | | | | GA-RCO | **83.20%** | +1.67% | 7215.669 | 128.143 |
| Madelon | 2600 | 500 | 2 | CART | 69.27% | | | (1.109) |
| | | | | GA-R | 61.29% | +6.15% | | 787.900 |
| | | | | GA-RC | 69.94% | +1.16% | 58.721 | 1859.012 |
| | | | | GA-RCO | **70.35%** | +1.45% | 7258.721 | 2671.267 |

⋆ The best performing solutions for each dataset are highlighted in **bold**.

⋆ Time in parenthesis shows elapse time of full CART in *rpart*.

**Table 7:** Full results at depth 5.

| DATASET | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Name | n | p | K | | Mean Out-Sample Accuracy | Mean In-Sample Accuracy Improvement | Pop. Gen. Time (sec) | GA Time (sec) |
| Wine | 173 | 13 | 3 | CART | 91.02% | | | (0.237) |
| | | | | GA-R | 88.96% | +2.31% | | 2.010 |
| | | | | GA-RC | **92.83%** | +1.43% | 0.469 | 4.753 |
| | | | | GA-RCO | 92.82% | +1.72% | 1812.069 | 5.907 |
| Chess | 1396 | 36 | 2 | CART | 89.86% | | | (0.427) |
| | | | | GA-R | 93.87% | +13.15% | | 1181.887 |
| | | | | GA-RC | 94.08% | +0.08% | 8.125 | 1022.221 |
| | | | | GA-RCO | **94.13%** | +0.11% | 7208.125 | 1551.750 |
| Image Segment. | 210 | 19 | 7 | CART | 77.14% | | | (0.253) |
| | | | | GA-R | 76.83% | +7.26% | | 3.464 |
| | | | | GA-RC | 82.15% | +1.41% | 0.684 | 5.955 |
| | | | | GA-RCO | **82.60%** | +1.85% | 7200.682 | 9.293 |
| Parkinson | 756 | 754 | 2 | CART | 80.95% | | | (0.767) |
| | | | | GA-R | 79.56% | +1.15% | | 42.843 |
| | | | | GA-RC | 83.54% | +2.05% | 15.850 | 152.228 |
| | | | | GA-RCO | **83.65%** | +2.24% | 7215.850 | 259.271 |
| Madelon | 2600 | 500 | 2 | CART | 69.27% | | | (1.164) |
| | | | | GA-R | 70.15% | +10.85% | | 2122.725 |
| | | | | GA-RC | 70.56% | +2.13% | 60.626 | 3973.067 |
| | | | | GA-RCO | **71.09%** | +2.73% | 7260.626 | 7535.475 |

⋆ The best performing solutions for each dataset are highlighted in **bold**.

⋆ Time in parenthesis shows elapse time of full CART in *rpart*.

R populations. In the other two datasets, GA with RC is the best one, however, there is no significant difference between RC and RCO populations' results. Only in one of them, GA with RC and GA with RCO show equal performance and there is no improvement over the initial best trees. No improvement means GA cannot improve provided initial trees, but this only happens in the Chess dataset at depth 2.

Chess dataset mostly includes binary feature values, then we noticed that our proposed algorithm cannot do a good job over binary features. However, there is an improvement at GA with R population, which shows that if initial trees are low quality (randomly generated) our GA can improve even if the dataset is binary. In higher depths, our GA can improve high-quality tress for chess dataset as well, but the improvement amount is very limited. We observe improvement in higher depths but there is no improvement in the depth 2. The reason is, in lower depths there is limited change potential in trees, especially binary features we just change related features for each split rule in each node because using different threshold has no effect over binary features.

Proposed GA shows 1% to 20% nominal improvement over pure CART at depth 2 depending on the datasets. Improvement amounts of GA with all population type increases at higher depths. Moreover, at higher depths, GA with the RCO population is the best one in general. Especially at depth 5 (see Table 7), for 4 out of 5 datasets, GA with RCO is always the best method. And in the last one, GA with RC and RCO show very close performances. At higher depths, the maximal tree includes more nodes, thus we have more diverse trees and more modification possibilities in crossover and mutation operations.

For more detail, we analyze the results of the Parkinson dataset at all depths. Parkinson dataset contains a moderate number of points, but it has the most feature among all datasets we use. Thus data size is not only dependent on the number of points, but also on the feature number. In this dataset, when depths are getting

deeper, out-sample accuracies are increased as in all other datasets. But, a more crucial point is the difference between the performances of GA with R, RC and RCO populations. At all depths, result of GA with RCO is always outperformed CART and other GA versions results. Also, in-sample accuracy improvements increase as the tree gets deeper, but the lowest increase is between depth 4 and 5. So, this does not mean that we get the best tree at depth >5 because each dataset has a tree with the fittest depth in terms of their sizes. If we use deeper trees than the proper depth, the tree becomes heavily dependent on the training set, and we might face with overfitting. As the tree at high depth contains more rules specified with training data, this results in low accuracy for test data. Therefore, our depth limit provides controlled growth for the data. And we will decide the best depth by examining each depth result.

To decide the best performing method, it is not enough to compare absolute accuracies, thus some additional statistical tests need to be considered. For that purpose, we analyze all datasets and all depths in together, so we test 20 different cases (4 datasets at 5 depths). We apply paired tests for each method to ensure that which method's performance is statistically significant. We use paired t-test if accuracy outputs show normal distribution, and Wilcoxon test if they are not normally distributed. For normality check, we use the Kolmogorov-Smirnov (KS) test because of the size. We test whole accuracy outputs for 100 replications and 5 folds; thus, our size is 500 for each case. Shapiro-Wilk is the most sensitive test for normality test but for large sizes ($> 50$) this is not powerful; thus, we choose KS test which is sensitive for large datasets. In the t-test and Wilcoxon test, we apply a one-tailed test to show selected methods average out-sample accuracy is significantly greater than the other method. So, we apply these tests with 0.05 critical value and for each method respectively. In the result table, inconclusive means, there is no evidence to say that any of these methods is significantly the best one in that case.

Thus, firstly we compare performances of RC and RCO population pairwise. We want to show that, adding sub-OCT trees provides an improvement over the performances. Thus we define alternative hypothesis as $\mu_{GA-RC} < \mu_{GA-RCO}$ where null hypothesis is $\mu_{GA-RC} = \mu_{GA-RCO}$. Some results in the above tables, cannot show a considerable difference between GA with RC and RCO population in terms of absolute accuracy. But when we analyze accuracy results with paired statistical tests for every 100 replications and each 5 fold against RC population and RCO population, results show generally RCO population's performance is significantly better than the RC population, but for some cases they are inconclusive. In more detail, for 0.05 critical value, out of 20 cases (4 different depths with 5 datasets), 7 of them show if we use the RCO population to initialize GA, it is significantly better than the RC population, 3 of them show GA with the RC population is better than RCO. The rest of the 10 cases show there is not any significant evidence to show the RC population or RCO population is the best one (see Table 8).

**Table 8:** Statistical test results when critical value = 0.05. Each value shows total number of wins/significantly better cases out of 20. Alternative Hypothesis is : $\mu_{GA-RC} < \mu_{GA-RCO}$

| **Method** | **# win** |
|---|---|
| *GA-RC* | 3 |
| *GA-RCO* | 7 |
| *Inconclusive* | 10 |

Secondly, we test GA with the RCO population, which is the winner of previous tests, against pure CART results. We compare these two methods, because our benchmark algorithm is CART and we suppose to outperform CART results with our approach. Thus, firstly we choose the best method we proposed and then we test our best method against our benchmark results. For the results of RCO population, we take out of sample accuracy results for every 100 replications and each 5 fold for every 20 cases, and we use CART results which are conducted with whole training data. Also, we define alternative hypothesis as $\mu_{CART} < \mu_{GA-RCO}$. Then these

results show that 19 cases out of 20, GA with RCO is always the best one against pure CART performance (see Table 9). So, this shows that GA can perform significant improvements over greedy CART trees.

**Table 9:** Statistical test results when critical value = 0.05. Each value shows total number of wins/significantly better cases out of 20. Alternative Hypothesis is : $\mu_{CART} < \mu_{GA-RCO}$

| **Method** | **# win** |
|---|---|
| *GA-RCO* | 19 |
| *CART* | 1 |

Besides statistical tests, box plots in Figure 9 show comparisons of GA with the RC population and RCO population for each dataset and at 4 different depths. Each box plot includes 500 out-sample accuracies (results of 5 folds, 100 replications). In these plots, especially chess dataset shows less or no difference between RC and RCO, as we have explained before. At higher depths, GA with the RCO population shows a slightly better performance.

With Statistical test and box plots, we just compare RC and RCO populations, because in this study we want to remark the contribution of GA to improve greedy results. Also, we would like to show augmenting the initial population with suboptimal trees beside greedy ones increase the output performance of GA. In other words, initialize GA with an intelligent population increases accuracy.

These results show that our GA can outperform CART in all datasets when we use an intelligent population (RC and RCO) rather than the solely random one (R population). GA with R population cannot outperform CART in some experiments. On the other hand, at the mean in-sample improvement perspective, GA with R population shows higher accuracy improvement over in-sample accuracies. R population involves only random trees and accuracies are very low. This improvement metric shows the effect of the GA over the initial population with the training set and proposed GA can increase the accuracy of these low qualified random trees dramatically.

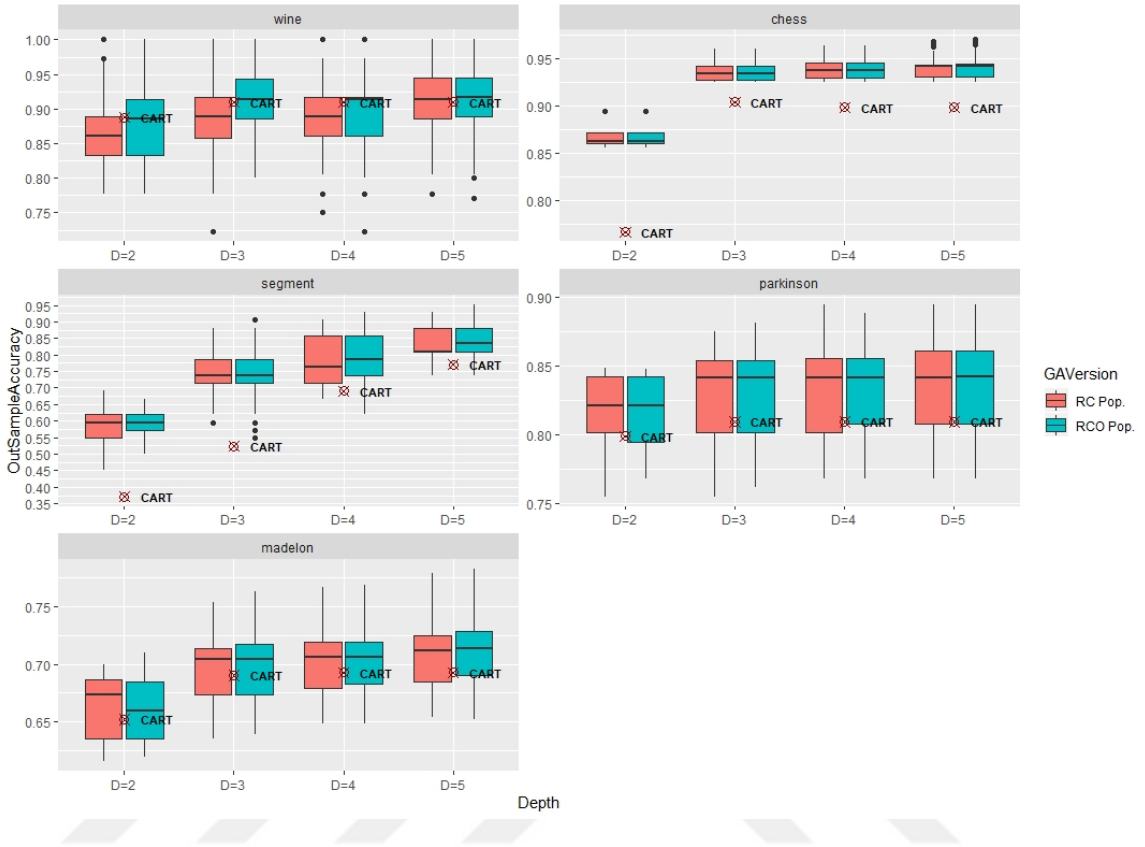


**Figure 9:** Effects of RC and RCO population on out-sample accuracies.

Finally, even if intelligent trees used in GA are not accurate enough, algorithm compose trees with higher prediction accuracy compared to the greedy CART algorithm for small, medium and large size datasets. With RC populations, GA execution takes reasonable time to achieve the final tree for small and medium-size datasets, also sub-CARTs are not resulting in a long time. For large size datasets (n>20,000), the execution time increases, especially when we increase the population size along with the data size. Also, construct a population with OCT costs too much time, but statistical tests show that sub-OCT trees also increase GA performances beside sub-CARTs. GA with the RC population also shows good improvement against CART, so it depends on your time sensitivity to work with RC or RCO population. Also, in all dataset sizes, we can observe at least 1% accuracy improvement, which is crucial in classification problems. If time is a big constraint, using the RC population will

provide sufficient results but if you need more accurate solutions and time is not a big challenge, then RCO population will be used to initialize proposed GA. Thanks to GA to produce high accuracy decision trees.

# CHAPTER VI

# CONCLUSION

In this thesis study, we propose an evolutionary algorithm, GA, for high accurate decision tree induction and we evaluate different variations. The aim of this work is to construct more accurate trees than greedy approaches and reach near-optimal solutions. Because of the disadvantages of greedy approaches and the complexity of optimal tree induction models, some heuristics will be fused to these methods to improve their performances. Hence, to improve greedy trees, CART, and to find near-optimal trees than the optimal classification tree model, OCT, in a shorter span of time, we apply a genetic algorithm. We construct the initial population of GA with sub-CART trees and/or sub-OCT trees. Random trees are inserted to the initial population as well to compare the performance of including CART and OCT sub-trees more clearly and to expand the search space with low performance solutions. We use CART accuracies as a benchmark to test our proposed algorithm's performance.

In the proposed study we evaluate different setups. In each setup, we define a maximum depth and restrict uncontrolled growth in a tree, and we try out different datasets with various sizes. Also, we have introduced 3 different initialization approach for GA. The first approach is, initializing GA with a fully random initial population. Random trees present poor quality, low accuracy, however, proposed GA improved their performances dramatically but cannot beat CART results at all setups (results with given dataset and depth). In the second approach, we add CART based full and sub-trees beside the random ones. We call this population as RC population. CART is a greedy approach; thus, we worked up to improve them with GA. Results

show GA with the RC population improves the performance of trees in the initial population and outperforms the CART result for a given setup. Although, optimal tree induction is a popular topic, but constructing an optimal binary split tree is complex and NP-Hard problem, especially with large size datasets. Thus, we divide datasets into subsets as we do to construct sub-CARTs and build sub-trees with optimization model OCT. Then, we add up this sub-OCTs besides the random and sub-CART ones and this tree blends called the RCO population. When we include CART and OCT solutions to the population together, we can improve their performances and outperform pure CART at all setups. Results show GA with RC and RCO initial populations are better than the R population and benchmark CART. Thus, we can generate better trees than the CART. Our algorithm may have little resemblance to the random forest (RF), but because of RF's less interpretable nature, our algorithm provides more explainable and interpretable solution.

For future work, we can apply some additional operators and steps to this proposed heuristic to observe more improvements and construct higher accurate trees. We can consider pruning steps at the end of the GA implementation. And, we can increase improvements with post-processing operations and simplify generated trees. Also, we can consider bais-variance tradeoff in the fitness function and we can consult to some other performance metrics, such as AUC, recall, precision and etc. to evaluate the performance of proposed work in more detail. And, we can compare the performance of our algorithm with Random Forest as well. Additionally, the optimal tree problem can be remodeled and new algorithms can be generated to classified large datasets optimally with less time cost.

This study shows that heuristic algorithms, such as GA, can improve the performances of greedy trees and also using potential of optimization models in GA provide powerful insights. Therefore, when we combine power of greedy trees and optimal trees we can construct high accuracy decision trees to classify future points robustly.

# Bibliography


[1] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[2] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[3] K. P. Bennett and J. A. Blue, "Optimal decision trees," *Rensselaer Polytechnic Institute Math Report*, vol. 214, p. 24, 1996.

[4] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, pp. 1039–1082, 2017.

[5] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[6] E. Kolçe and N. Frasheri, "The use of heuristics in decision tree learning optimization," *International Journal of Computer Engineering in Research Trends*, vol. 1, no. 3, pp. 127–130, 2014.

[7] J. Bala, J. Huang, H. Vafaie, K. DeJong, and H. Wechsler, "Hybrid learning using genetic algorithms and decision trees for pattern classification," in *IJCAI (1)*, pp. 719–724, 1995.

[8] A. Papagelis and D. Kalles, "Ga tree: genetically evolved decision trees," in *Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000*, pp. 203–206, IEEE, 2000.

[9] D. Jankowski and K. Jackowski, "Evolutionary algorithm for decision tree induction," in *IFIP International Conference on Computer Information Systems and Industrial Management*, pp. 23–32, Springer, 2015.

[10] Z. Fu, B. L. Golden, S. Lele, S. Raghavan, and E. A. Wasil, "A genetic algorithm-based approach for building accurate decision trees," *INFORMS Journal on Computing*, vol. 15, no. 1, pp. 3–22, 2003.

[11] M. Ryan and V. Rayward-Smith, "The evolution of decision trees," in *Proceedings of the Third Annual Conference on Genetic Programming*, pp. 350–358, San Francisco, CA.: Morgan Kaufmann, 1998.

[12] J. N. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal," *Journal of the American statistical association*, vol. 58, no. 302, pp. 415–434, 1963.

[13] R. Messenger and L. Mandell, "A modal search technique for predictive nominal scale multivariate analysis," *Journal of the American statistical association*, vol. 67, no. 340, pp. 768–772, 1972.

[14] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 29, no. 2, pp. 119–127, 1980.

[15] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[16] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers-a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005.

[17] B. Kamiński, M. Jakubczyk, and P. Szufel, "A framework for sensitivity analysis of decision trees," *Central European journal of operations research*, vol. 26, no. 1, pp. 135–159, 2018.

[18] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information processing letters*, vol. 5, no. 1, pp. 15–17, 1976.

[19] T. Hancock, T. Jiang, M. Li, and J. Tromp, "Lower bounds on learning decision lists and trees," *Information and Computation*, vol. 126, no. 2, pp. 114–122, 1996.

[20] H. Zantema and H. L. Bodlaender, *Finding small equivalent decision trees is hard*, vol. 1999. Utrecht University: Information and Computing Sciences, 1999.

[21] G. Naumov, "Np-completeness of problems of construction of optimal decision trees," *SPhD*, vol. 36, p. 270, 1991.

[22] M. Bala and R. Agrawal, "Optimal decision tree based multi-class support vector machine," *Informatica*, vol. 35, no. 2, 2011.

[23] D. Bertsimas and R. Shioda, "Classification and regression via integer optimization," *Operations Research*, vol. 55, no. 2, pp. 252–271, 2007.

[24] S. Verwer and Y. Zhang, "Learning decision trees with flexible constraints and objectives using integer optimization," in *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 94–103, Springer, 2017.

[25] O. Gunluk, J. Kalagnanam, M. Li, M. Menickelly, and K. Scheinberg, "Optimal generalized decision trees via integer programming," *arXiv preprint arXiv:1612.03225*, 2016.

[26] T. N. Phyu, "Survey of classification techniques in data mining," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2009.

[27] Q. Zhao and M. Shirasaka, "A study on evolutionary design of binary decision trees," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 3, pp. 1988–1993, IEEE, 1999.

[28] B.-B. Chai, X. Zhuang, Y. Zhao, and J. Sklansky, "Binary linear decision tree with genetic algorithm," in *Proceedings of 13th International Conference on Pattern Recognition*, vol. 4, pp. 530–534, IEEE, 1996.

[29] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh, "Boatoptimistic decision tree construction," in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pp. 169–180, 1999.

[30] D. E. Goldberg, "Genetic algorithms in search," *Optimization, and Machine-Learning*, 1989.

[31] M. Mitchell, "An introduction to genetic algorithms, massachusetts institute of technology," 1996.

[32] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[33] S.-H. Cha and C. C. Tappert, "A genetic algorithm for constructing compact binary decision trees," *Journal of pattern recognition research*, vol. 4, no. 1, pp. 1–13, 2009.

[34] K. M. Kim, J. J. Park, M. H. Song, I. C. Kim, and C. Y. Suen, "Binary decision tree using genetic algorithm for recognizing defect patterns of cold mill strip," in *Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 461–466, Springer, 2004.

[35] S. P. Teeuwsen, I. Erlich, M. A. El-Sharkawi, and U. Bachmann, "Genetic algorithm and decision tree-based oscillatory stability assessment," *IEEE Transactions on Power Systems*, vol. 21, no. 2, pp. 746–753, 2006.

[36] A. Alhindi, "Optimizing training data selection for decision trees using genetic algorithms," *IJCSNS*, vol. 20, no. 4, p. 84, 2020.

[37] Z. Fu, "An innovative ga-based decision tree classifier in large scale data mining," in *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 348–353, Springer, 1999.

[38] K. Bache and M. Lichman, "Uci machine learning repository [http://archive. ics. uci. edu/ml]. irvine, ca: University of california," *School of information and computer science*, vol. 28, 2013.

[39] T. Therneau and B. Atkinson, *rpart: Recursive Partitioning and Regression Trees*, 2019. R package version 4.1-15.

[40] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[41] X. Nodet, "Cplex optimization studio 12.10 is available," Jul 2020.

# VITA

Elif ERSOY graduated from Beşiktaş Atatürk Anatolian High School in 2013. She received her B.S. degree in industrial Engineering from Özyeğin University in June 2017. After graduation, she joined Master of Science program in Industrial Engineering at Özyeğin University and has been working under supervision of Asst. Prof. Enis Kayış and Asst. Prof. Erinç Albey. She has worked as a teaching assistant and assists Asst. Prof. Ümmühan Akbay in course of Operations Research II. Her research mainly focuses on data science.