Knowledge Representation Using Decision Trees Constructed Based on Binary Splits

Mohammad Azad*
College of Computer and Information Sciences,
Jouf University
Sakaka, 72441 – Saudi Arbia
[e-mail: mmazad@ju.edu.sa]
*Corresponding author: Mohammad Azad

Received January 8, 2020; revised June 10, 2019; accepted August 15, 2020; published October 30, 2020

Abstract

It is tremendously important to construct decision trees to use as a tool for knowledge representation from a given decision table. However, the usual algorithms may split the decision table based on each value, which is not efficient for numerical attributes. The methodology of this paper is to split the given decision table into binary groups as like the CART algorithm, that uses binary split to work for both categorical and numerical attributes. The difference is that it uses split for each attribute established by the directed acyclic graph in a dynamic programming fashion whereas, the CART uses binary split among all considered attributes in a greedy fashion. The aim of this paper is to study the effect of binary splits in comparison with each value splits when building the decision trees. Such effect can be studied by comparing the number of nodes, local and global misclassification rate among the constructed decision trees based on three proposed algorithms.

Keywords: Knowledge representation, decision trees, binary split, directed acyclic graph

The author would like to express the gratitude to Jouf University for the support of my research and give thanks to Mikhail Moshkov for stimulative discussion and suggestion. The author also greatly appreciative to the anonymous reviewers for useful remarks to improve the quality of the paper.

http://doi.org/10.3837/tiis.2020.10.005 **ISSN : 1976-7277**

1. Introduction

Decision trees can be used as a tool for knowledge representation that store information from the given data table to solve problems in many domains including decision support systems, machine learning, data mining, etc. Decision trees can be considered for optimization of its parameters for various purposes. For example, when we consider the optimization of the depth of tree, we match it with worst-case time complexity of algorithm (by simulating the tree as the work of the algorithm). Similarly, we consider the optimization of the average depth and number of nodes of tree by matching the average time complexity and space complexity of algorithm. Therefore, it is very natural to not only consider single criterion but also bi-objective optimization (BOO) of such different parameters. In the case of BOO, it is necessary to study number of misclassification (#misclassifications) vs. number of nodes so that the optimized tree can be useful for both prediction and well as shorter trees for knowledge representation.

In this paper, the aim is to study BOO by comparing the following three cost functions of the tree *TR* for a decision table (data set) *TB* (shown below in **Table 1**).

Table 1. Description of the three cost functions under study

Cost function	**Description**
NN(TR)	the number of nodes in *TR* for the table *TB*
GM(TB, TR)	the global misclassification rate of *TR* for the table *TB*
LM(TB, TR)	the local misclassification rate of *TR* for the table *TB*

The cost functions *GM(TB, TR)* and *LM(TB, TR)* has been mentioned first in [3]. *GM(TB, TR)* is identical to the #misclassifications of *TR* on *TB* divided by the number of rows (#rows) in *TB*. *LM(TB, TR)* is the maximum of ($\frac{\#misclassifications}{\#\,rows}$) among all terminal nodes of *TR* on *TB*. It is easy to manifest that *GM(TB, TR)* is at most *LM(TB, TR)*

As it is stated before that, the goal is to find a decision tree *TR* that has smaller number of nodes as well as smaller # misclassifications. In [3], it is found that the global misclassification rate may not be efficient cost function because #misclassifications may not be equally dispersed. Besides, the proportion of #misclassifications can be big enough for a certain terminal nodes. Therefore, the local misclassification rate is also very important.

Let us review the literature regarding the optimization associated with decision trees. Many of such problems are NP-hard [9, 10, 23]. Nonetheless, many researchers have studied approximate optimization techniques for decision trees (for example, genetic algorithms [15], simulated annealing [17], and ant colony techniques [12]). Besides, other researchers have studied the comprehensive algorithms for decision tree optimization (for example, brute-force algorithms [19], dynamic programming [16, 18, 20], and branch-and-bound technique [14]). However, none has studied in the direction of dynamic programming for bi-objective (and multi-objective) optimization of decision trees except the authors of [1, 2]. Previously, the authors [3] proposed three algorithms for BOO of decision tree based on expansion of dynamic programming and studied the parameters *NN*, *GM*, and *LM* of the constructed decision trees. Unfortunately, these algorithms are applicable to decision tables with

categorical attributes only. Another problem is that sometimes, the number of nodes in the constructed decision trees is very large.

In this paper, two algorithms (*GM* and *LM* algorithms) have been designed for the construction of decision trees, that can be suitable to middle-sized tables containing categorical as well as numerical attributes. These algorithms are based on expansions of dynamic programming — BOO of CART-like decision trees (CL-trees) [1, 2, 6, 13] relative to the parameters NN and GM, and relative to the parameters NN and LM. In CL-trees, instead of the initial attributes, binary splits have been used as it was done in the original CART algorithm [13]. One more algorithm (GLM algorithm) is also designed which is the mixture of the GM and LM algorithm. The considered algorithms have been applied to 14 decision tables from the UCI Machine Learning Repository [4], and three parameters are studied NN, GM, and LM of the constructed trees. Furthermore, the obtained results are compared with the previous results in [3].

The obtained results show that the new algorithms produce decision trees containing on average smaller number of nodes, smaller global misclassification rate as well as smaller local misclassification rate for many decision tables. Therefore, the consideration of such algorithms can be useful for the extraction of knowledge from middle-sized decision tables and for its representation by decision trees. These algorithms can be used in different areas of data analysis including rough set theory [21, 22].

The rest of the paper is organized as follows. In Sect. 2, the methodology of the algorithms is explained. In Sect. 3, experimental setup and results are discussed. In Sect. 4, the outcome of the experiments is analyzed. Section 5 contains short conclusions.

2. Methodology

In literature, there are many ways to construct decision trees i.e., ant colony algorithms, genetic algorithms, greedy algorithms, branch and bound techniques but an expansion of dynamic programming is used in this paper, which gives all possible decision trees under consideration. In this section, the construction of directed acyclic graph (DAG) has been described first and then, the extraction of decision trees is demonstrated, then the BOO is explained and finally, the three algorithms are described.

2.1 Building of Decision Trees

The algorithm (described in [1, 2, 6]) is used for the construction of a DAG based on the expansion of the dynamic programming (DP). Normally, the goal of the DP is to find an optimal solution. Using this expansion of DP, a DAG is constructed, which describes all decision trees for the decision table under consideration, and later, based on this DAG, another algorithm is considered for the BOO of decision trees for the decision table under consideration relative to two different parameters. For example, a decision table *TB* is shown in **Table 2** and corresponding DAG in **Fig. 1**.

Table 2. An example of a decision table *TB*

f_1	f_2	
0	0	1
0	1	3
1	0	1
1	1	2

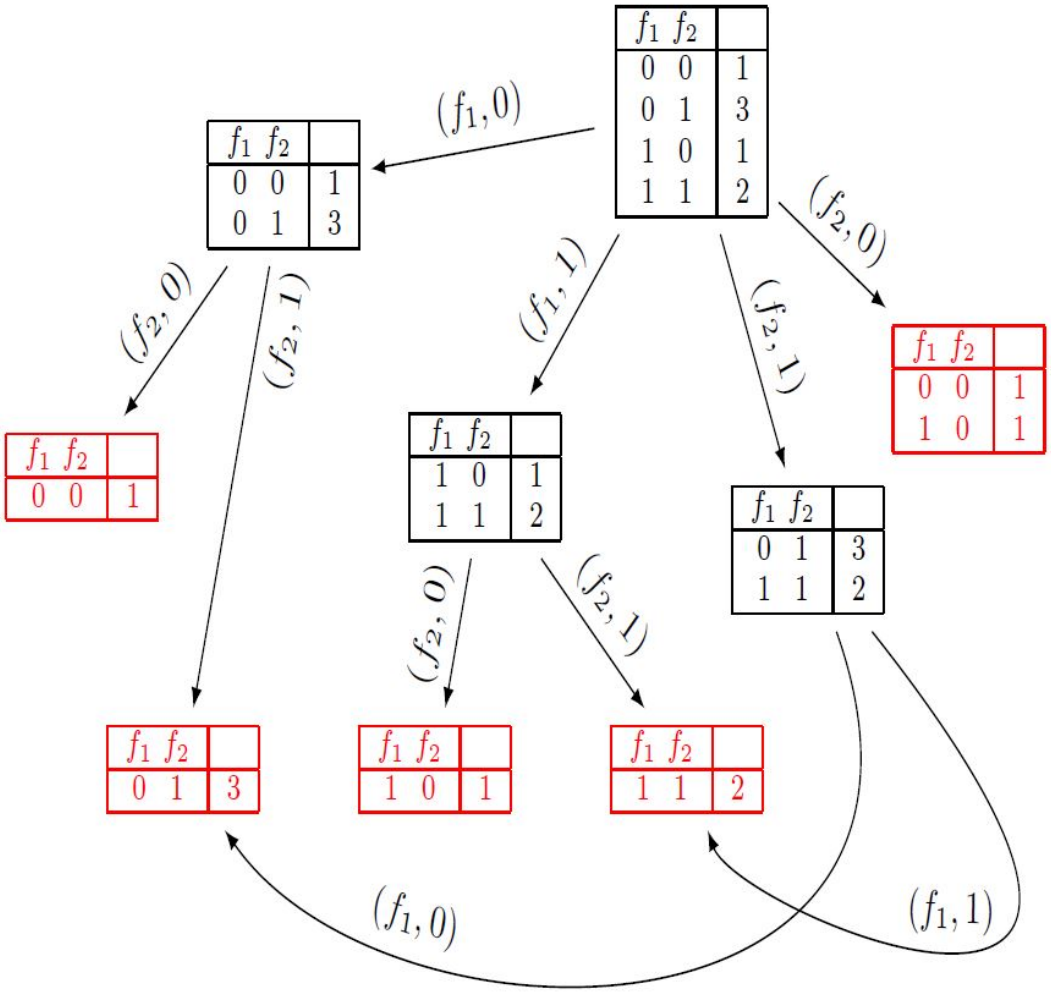


Fig. 1. The DAG for the table *TB*

After constructing the DAG, all decision trees for the table under consideration can be found. **Fig. 2** shows the two possible comprehensive decision trees (the first one by the root f_1 and the second tree by the root f_2) for the table TB in **Table 2**. Even though the time complexity of the construction of the DAG is exponential in the worst case depending on the size of the table, it is still possible to use this method for small to middle-sized decision tables [1, 2, 6].

There are many ways to split or partition the tables into subtables. Each value splits are considered in paper [3] which is applicable for categorical attributes only. In this paper, binary splits are considered as like in CART [13]. These trees are called CL-trees. This technique is suitable for decision tables containing numerical as well as categorical attributes.

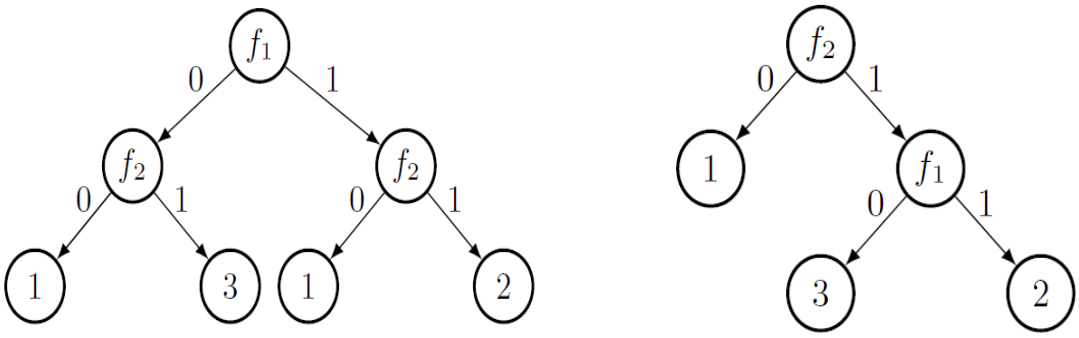


Fig. 2. Two decision trees derived from the DAG in **Fig. 1**

2.2 CL-Trees

For CL-trees, an optimal partition is used for some attributes at each non-terminal vertex. But, the standard CART algorithm utilizes an optimal partition for all considered attributes at each non-terminal vertex. Based on this approach, a larger collection of decision trees can be obtained.

Let us consider $f_1, \ldots, f_n$ attributes (either numerical or categorical) and decision attribute i.e., label (categorical) for a given decision table *TB*. Each attribute f_i $(1 \leq i \leq n)$ is converted into binary attribute (as in CART) using binary partition.

In case if the attribute f_i is categorical, the set of its' values B is partitioned into two nonempty subsets B_1 and B_0. The partition t is considered as *0* if the set B_0 contains the f_i's value and *1* otherwise. But if the attribute f_i is numerical, then it is converted to a binary attribute by comparing with a real threshold α. The partition t is considered as *0* if f_i's value is smaller compare to α, and *1* otherwise. As a result of the partition t, two subtables $TB_{t=0}$ and $TB_{t=1}$ are obtained.

Gini index was used as an uncertainty parameter (U) in [1], whereas, in this paper "abs" [2] is used as an uncertainty parameter (U). The impurity function is considered as a quality index for partition. Let assume after partition t for the table *TB*, two subtables $TB_{t=0}$ and $TB_{t=1}$ are obtained. Then, the impurity function, *I(TB, t)*, can be calculated by the weighted sum of U of two such subtables (note that the weights are in proportion to the total objects (rows) in the corresponding subtables). If the partition t is applied for the attribute f_i in *TB* that produces minimum impurity function *I(TB, t)*, then this partition is called the best partition for the attribute f_i.

Every terminal vertex is tagged by a decision attribute i.e., label in CL-trees for *TB*. Every non-terminal vertex is tagged by a partition based on one of the attributes. Also, two arcs are departing from this non-terminal vertex (one is labeled with *0* and another with *1*). Let assume TR is a decision tree and v is a vertex of it. Then, a subtable $TB(TR, v)$ for the given table *TB* can be mapped for each vertex v. This subtable $TB(TR, v)$ comprises all objects (rows) of the given table *TB* for which the work of the decision tree will be carried out on the vertex v. Our tree is built upon a few assumptions:

1. For each non-terminal vertex v, $TB(TR, v)$ comprises objects (rows) with non-identical labels. Furthermore, the vertex v is tagged by a best split for a non-constant attribute f_i on $TB(TR, v)$.
2. For each terminal vertex v, the vertex is tagged with a most frequent label. This is a decision which is bound with the largest amount of objects (rows) in the corresponding subtable.

2.3 Bi-objective Optimization (BOO) and Pareto Fronts

BOO considers the optimization of two objective functions simultaneously. In this paper, the idea is to optimize both number of nodes and number of misclassification of the tree simultaneously. For this purpose, the algorithm A_{POPs} [1, 2, 6] has been used.

Let us consider a point (r, s) in X (where X is a finite set of points in 2D space). This point is defined as a Pareto optimal point (POP) for X if no point (p, q) in X (where $p \leq r$ and $q \leq s$) such that $(r, s) \neq (p, q)$ [1, 2, 6]. A sample example is given in **Fig. 3** for illustration purposes. The collection of POPs for a particular problem is called the Pareto front of the problem.

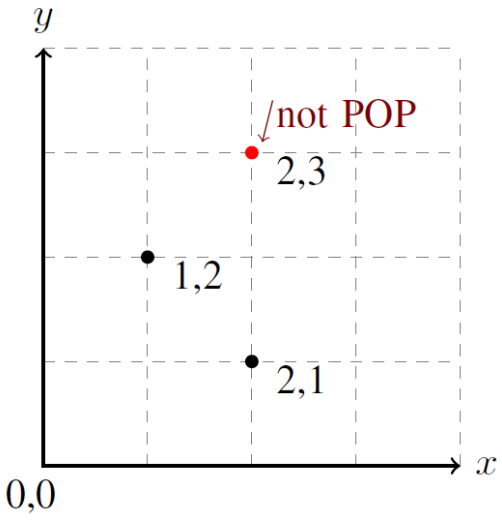
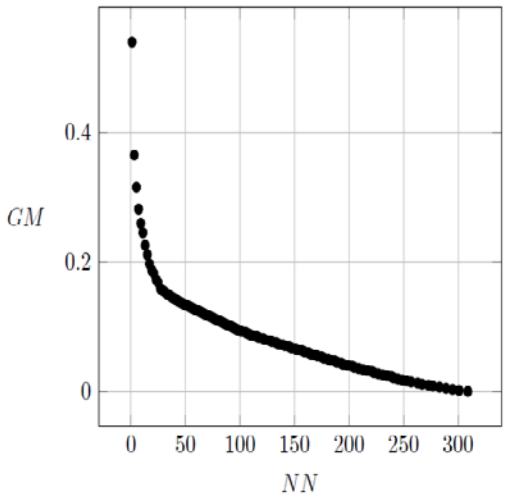


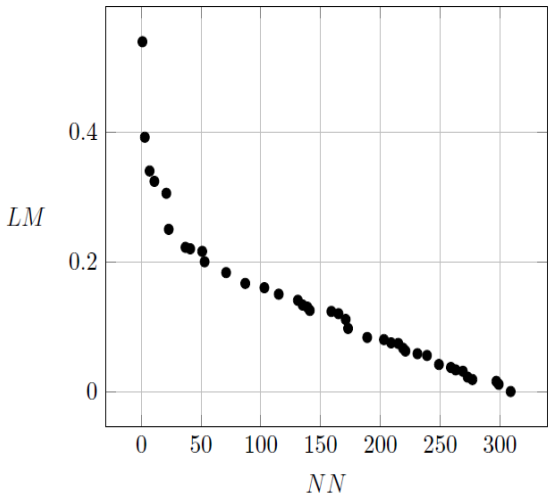
Fig. 3. An example of Pareto front in 2D space

2.4 Three Algorithms

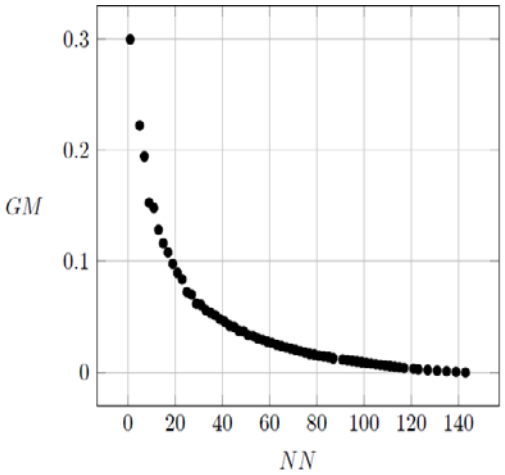
The main idea is to use the algorithm A_{POPs} [1, 2, 6] which, for a given decision table, constructs the collection of POPs for the problem of BOO of CL-trees proportionate to the parameters NN and GM (see, for example, **Fig. 4** (a), (c), (e)). This algorithm can be expanded to the construction of the collection of POPs for the problem of BOO of CL-trees proportionate to the parameters NN and LM (see, for example, **Fig. 4** (b), (d), (f)). For each POP, a decision tree can be derived with values of the considered parameters equal to the coordinates of this point.



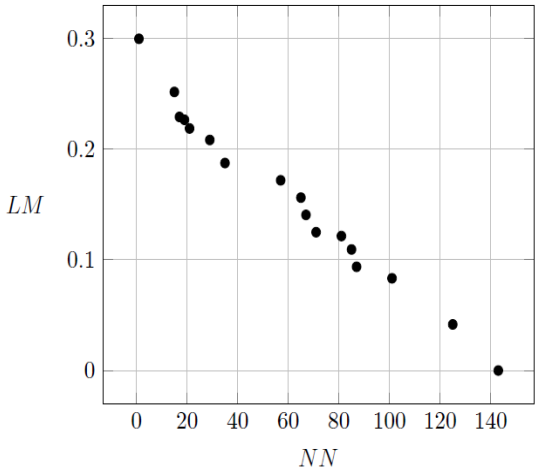
(a) BALANCE-SCALE, NN and GM



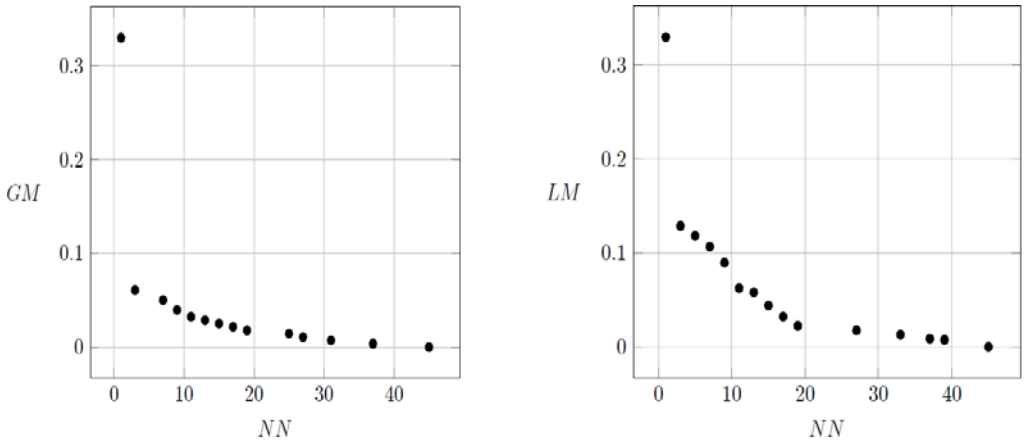
(b) BALANCE-SCALE, NN and LM



(c) CARS, NN and GM



(d) CARS, NN and LM



(e) HOUSE-VOTES, NN and GM (f) HOUSE-VOTES, NN and LM

Fig. 4. Pareto front for decision tables BALANCE-SCALE, CARS, and HOUSE-VOTES for pairs of parameters NN, GM and NN, LM

Below three algorithms for decision tree construction are described based on the use of the algorithm A_{POPs} and its expansion which time complexity in the worst case is exponential. Consequently, the complexity of the below algorithms in the worst case is exponential as well.

2.4.1 GM Algorithm

For a given decision table *TB*, the set of POPs is constructed using the algorithm A_{POPs} for the problem of BOO of CL-trees proportionate to the parameters NN and GM. The coordinates of POPs are normalized: for each POP, the value of each coordinate is divided by the highest value (among all POPs) of the corresponding coordinate. After that, a normalized POP is chosen with the smallest distance (Euclidean distance) from the (0, 0) point. The coordinates of this point are restored and a decision tree *TR* is derived, for which the values of the parameters *NN* and *GM* are equal to the restored coordinates. The tree *TR* is the output of *GM* algorithm. Below is the pseudo code of the *GM* algorithm.

GM Algorithm : Input: *TB*, Output: *TR*

1. Begin
2. Construct DAG for *TB* and Initialize, *S*, the set of POPs by using the Algorithm A_{POPs} for BOO of CL-trees proportionate to the parameter *NN* and *GM*
3. Normalize the coordinates of *S*
4. $Distance_{min}$ = -1 *//Store the nearest distance to (0, 0)*
5. *P* ←*(0, 0)// Store the point nearest to (0, 0)*
6. For each point (a, b) from S
7. *Distance* ←Find the Euclidean distance from the origin (0, 0)
8. If (*Distance* < $Distance_{min}$) // update the distance and point P
9. $Distance_{min}$ = *Distance*
10. *P* ←*(a, b)*

11. End If
12. End For
13. Derive a decision tree *TR* from the DAG for the point *P*
14. Return *TR*
15. End

2.4.2 LM Algorithm

This algorithm works in the same way as *GM* algorithm but instead of the parameters *NN* and *GM* it uses the parameters *NN* and *LM*. For a given decision table *TB*, the set of POPs is constructed using the expansion of the algorithm A_{POPs} for the problem of BOO of CL-trees proportionate to the parameters *NN* and *LM*. Below is the pseudo code of the *LM* algorithm.

LM Algorithm : Input: *TB*, Output: *TR*

1. Begin
2. Construct DAG for *TB* and Initialize, *S*, the set of POPs by using the Algorithm A_{POPs} for BOO of CL-trees proportionate to the parameter *NN* and *LM*
3. Normalize the coordinates of *S*
4. $Distance_{min}$ = -1 *//Store the nearest distance to (0, 0)*
5. *P* ←*(0, 0)// Store the point nearest to (0, 0)*
6. For each point (a, b) from S
7. *Distance* ←Find the Euclidean distance from the origin (0, 0)
8. If (*Distance* < $Distance_{min}$) // update the distance and point *P*
9. $Distance_{min}$ = *Distance*
10. *P* ←*(a, b)*
11. End If
12. End For
13. Derive a decision tree *TR* from the DAG for the point *P*
14. Return *TR*
15. *End*

2.4.3 GLM Algorithm

First, the *GM* algorithm is applied to a given decision table *TB* and a decision tree TR_1 is built. After that, using the algorithm A_{POPs} the set of POPs is constructed for the parameters *NN* and *LM*. Then, a POP is chosen for which the value of the coordinate *NN* is closest to $NN(TR_1)$. At the end, a decision tree TR_2 is derived for which the values of the parameters *NN* and *LM* are equal to the coordinates of the chosen POP. The tree TR_2 is the output of *GLM* algorithm. Below is the pseudo code of the *GLM* algorithm.

GLM Algorithm : Input: *TB*, Output: *TR*

1. Begin
2. Apply GM algorithm, $TR_1 \leftarrow GM(TB)$
3. $NN_1 \leftarrow NN(TR_1)$
4. Construct DAG for *TB* and Initialize, *S*, the set of POPs by using the Algorithm A_{POPs} for BOO of CL-trees proportionate to the parameter *NN* and *LM*
5. $Distance_{min} = -1$ *//Store the nearest distance to* NN_1
6. $P \leftarrow (0, 0)$ *// Store the point nearest to* NN_1
7. For each point (a, b) from S
8. *Distance* ←Find the Euclidean distance from *a* to NN_1
9. If ($Distance < Distance_{min}$) // update the distance and point *P*
10. $Distance_{min} = Distance$
11. $P \leftarrow (a, b)$
12. End If
13. End For
14. Derive a decision tree *TR* from the DAG for the point *P*
15. Return *TR*
16. *End*

3. Experimental Evaluation

3.1 Data sets

The experiments have been performed on 14 decision tables from the UCI Machine Learning Repository [4] as portrayed in **Table 3**. The first column shows the name of the data set, the second column describes the number of rows, and the third column describes the number of attributes for the corresponding data set.

Table 3. Data sets

Name	Rows	Attributes
BALANCE-SCALE	625	5
BREAST-CANCER	266	10
CARS	1728	7
HAYES-ROTH	69	5
HOUSE-VOTES	279	17
IRIS	150	5
LENSES	1	5
LYMPHOGRAPHY	148	19
NURSERY	12960	9
SHUTTLE-LANDING	15	7
SOYBEAN-SMALL	47	36
SPECT-TEST	169	23

TIC-TAC-TOE	958	10
ZOO	59	17

3.2 Devices configuration

The experiments were mainly performed on a Windows machine that has the following configurations:

- Windows 7 operating system
- Intel core processor
- 128 GB RAM

The DAGGER [5] software system has been used that is written in C++ programming language. It uses both parallel threads and processors using MPI to accomplish the tasks.

3.3 Preprocessing of data sets

It is necessary and essential to preprocess the decision tables. The first problem is having the missing values. It has been solved by changing each missing values with the value of the feature that is most frequently used. The second problem is having the conditional attributes that take singular value for each row. It is necessary to remove such attributes to continue the experiments.

3.4 Experimental results

For each considered data set, the three algorithms (*GM* algorithm, *LM* algorithm, and *GLM* algorithm) are applied to get the values of *NN*, *GM*, and *LM* of decision trees. The experimental results are depicted in the **Table 4**. First column shows the name of the data set, then the second column shows the results of GM algorithm, which is partitioned into three groups: the value of *NN*, then the value of *GM*, and finally the value of *LM*. The same has been shown for the *LM* and *GLM* algorithm.

Table 4. Experimental results of three algorithms based on binary split

Data Set	**GM algorithm**			**LM algorithm**			**GLM algorithm**		
	NN	**GM**	**LM**	**NN**	**GM**	**LM**	**NN**	**GM**	**LM**
BALANCE-SCALE	43	0.14	0.5	53	0.16	0.2	41	0.1	0.22
BREAST-CANCER	37	0.11	0.29	31	0.14	0.15	37	0.12	0.15
CARS	29	0.06	0.34	71	0.05	0.13	29	0.14	0.21
HAYES-ROTH	9	0.19	0.33	17	0.06	0.17	9	0.19	0.33
HOUSE-VOTES	3	0.06	0.13	11	0.03	0.06	3	0.06	0.13
IRIS	5	0.04	0.09	5	0.04	0.09	5	0.04	0.06
LENSES	5	0.1	0.5	7	0	0	7	0	0
LYMPHOGRAPHY	9	0.15	0.2	9	0.16	0.17	9	0.16	0.17
NURSERY	37	0.08	0.34	35	0.1	0.17	35	0.1	0.17
SHUTTLE-LANDING	5	0.2	0.25	7	0.13	0.18	5	0.2	0.25
SOYBEAN-SMALL	5	0.21	0.37	5	0.21	0.32	5	0.21	0.32
SPECT-TEST	17	0.02	0.07	19	0.02	0.02	17	0.02	0.03
TIC-TAC-TOE	33	0.09	0.39	41	0.08	0.14	33	0.16	0.21
ZOO	7	0.25	0.43	9	0.19	0.28	7	0.27	0.42
Average	**17.43**	**0.12**	**0.3**	**22.86**	**0.1**	**0.15**	**17.29**	**0.13**	**0.19**
SD	**14.87**	**0.07**	**0.14**	**20.54**	**0.07**	**0.09**	**14.29**	**0.08**	**0.12**

At the end of the **Table 4**, the average and standard deviation (SD) are displayed.

In **Table 5**, the amount of improvement compared to the previous results [3] are shown. Here, '-' shows the reduction or improvement of the parameter and '+' shows the increment or non-improvement of the parameter. At the end of the table, the averaged and SD are displayed as well.

Table 5. The improvement of results when applying binary splits compared to each value splits (- shows the reduction and + shows the increment)

Data Set	GM algorithm			LM algorithm			GLM algorithm		
	NN	GM	LM	NN	GM	LM	NN	GM	LM
BALANCE-SCALE	-63	-0.04	+0.1	-133	+0.02	-0.04	-50	-0.12	-0.1
BREAST-CANCER	-22	0.0	-0.04	-27	+0.01	-0.01	-21	-0.01	-0.01
CARS	-69	-0.02	-0.16	-267	+0.04	+0.05	-106	0.0	-0.08
HAYES-ROTH	-14	+0.03	-0.17	-9	-0.07	-0.16	-17	+0.06	0.0
HOUSE-VOTES	0	0.0	+0.01	0	0.0	0.0	0	0.0	0.0
IRIS	0	0.0	0.0	0	0.0	0.0	0	0.0	-0.03
LENSES	-1	0.0	0.0	-1	0.0	0	-1	0	0
LYMPHOGRAPHY	-4	+0.02	-0.13	-2	0.0	-0.03	-2	0.0	-0.03
NURSERY	-37	0.0	0.0	-80	+0.01	-0.05	-35	0.0	-0.06
SHUTTLE-LANDING	-2	0.0	-0.08	+2	-0.14	-0.13	0	-0.07	-0.06
SOYBEAN-SMALL	+2	-0.21	-0.18	+2	-0.22	-0.18	+2	-0.22	-0.18
SPECT-TEST	0	0.0	-0.03	0	0.0	0.0	0	0.0	0.0
TIC-TAC-TOE	-39	-0.02	-0.07	-41	-0.04	-0.05	-39	+0.01	+0.01
ZOO	-1	+0.06	0.0	0	-0.01	0.0	-2	+0.07	+0.14
Average	**-17.86**	**-0.01**	**-0.06**	**-39.71**	**-0.03**	**-0.04**	**-19.35**	**-0.02**	**-0.03**
SD	**24.59**	**0.06**	**0.08**	**76.36**	**0.07**	**0.07**	**30.36**	**0.07**	**0.07**

4. Discussion

The results demonstrate the clear advantage of using the binary splits compared to the each value splits in [3]. The number of nodes in the tree is smaller in this experiment, also the accuracy is reasonable. For example, for the data set "BALANCE-SCALE" and GM algorithm, the number of nodes was 106 for each-value splits but now it is only 43, the global error-rate was 0.18 but now it is only 0.14 even though the local error-rate is slightly increased from 0.4 to 0.5. As for the LM algorithm, the number of nodes was 186 for each-value splits but now it is only 53, the global error-rate was 0.14 but now it is a bit high 0.18 and the local error-rate has been decreased from 0.24 to 0.2. As for the GLM algorithm, the number of nodes was 91 for each-value splits but now it is only 41, the global error-rate was 0.22 but now it is much less 0.1 and the local error-rate has been decreased from 0.32 to 0.22.

Another example that can be highlighted is the decision table "CARS". For this case with GM algorithm, the number of nodes was 98 for each-value splits but now it is only 29, the global error-rate was 0.08 but now it is only 0.06 and the local error-rate is also decreased from 0.5 to 0.34. As for the LM algorithm, the number of nodes was 338 for each-value splits but now it is only 71, the global error-rate was 0.01 but now it is a bit high 0.05, and the local error-rate has been increased from 0.08 to 0.13. As for the GLM algorithm, the number of nodes was 135 for each-value splits but now it is only 29, the global error-rate remains the same but the local error-rate has been decreased from 0.29 to 0.21.

Finally, the average of all decision tables for each combination has been calculated. For GM algorithm, the average number of nodes was before 35.29, which is now 17.43, the average GM was before 0.13, which is now 0.12, the average LM was before 0.36, which is now 0.3. For LM algorithm, the average number of nodes was before 62.57, which is now 22.86, the average GM was before 0.13, which is now 0.1, the average LM was before 0.19, which is now 0.15. For GLM algorithm, the average number of nodes was before 36.64, which is now 17.29, the average GM was before 0.15, which is now 0.13, the average LM was before 0.22, which is now 0.19.

It is possible to compare among the three algorithms built upon binary split. It is clear that that GLM algorithm have smaller NN, and competitive values of GM and LM.

To compare the methods statistically, Friedman test with the corresponding Nemenyi post-hoc test as suggested in [24] has been employed. Let us consider M decision tables TB_1, … , TB_M and k methods A_1, … , A_k. For each corresponding decision table, we rank the methods $A_1, \dots, A_k$ based on their performance scores (the value of NN, GM or LM), where we assign the best performing method the rank 1, the next one rank 2, and so on (in case of tie, the average is taken). After that, the average of ranks for each method is calculated over the M decision tables.

The evaluation of two methods is considered significantly different (assume a fixed significance level of α), when the corresponding average ranks is greater than the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6M}}$$

where q_α is a critical value (depending on α and k) for the two-tailed Nemenyi test [24].

In this work, the statistical tests are performed between binary and each value split for each algorithm. **Fig. 5,6,7**. shows the CDD (Critical Difference Diagram) for significance level of α = 0.05. It illustrates the average rank for each method on the x-axis. In this diagram, if no significant variation among the considered methods is observed by the Nemenyi test, then the methods are clustered by a line.

From **Fig. 5**, it is clear that, the best ranked algorithms are achieved by using the binary split when we minimize the NN parameter of the decision tree. Furthermore, for the case of GM and GLM algorithms (**Fig. 5(a) and (c)**), the binary split are stastistically significant (different) than the counterpart of each value split.

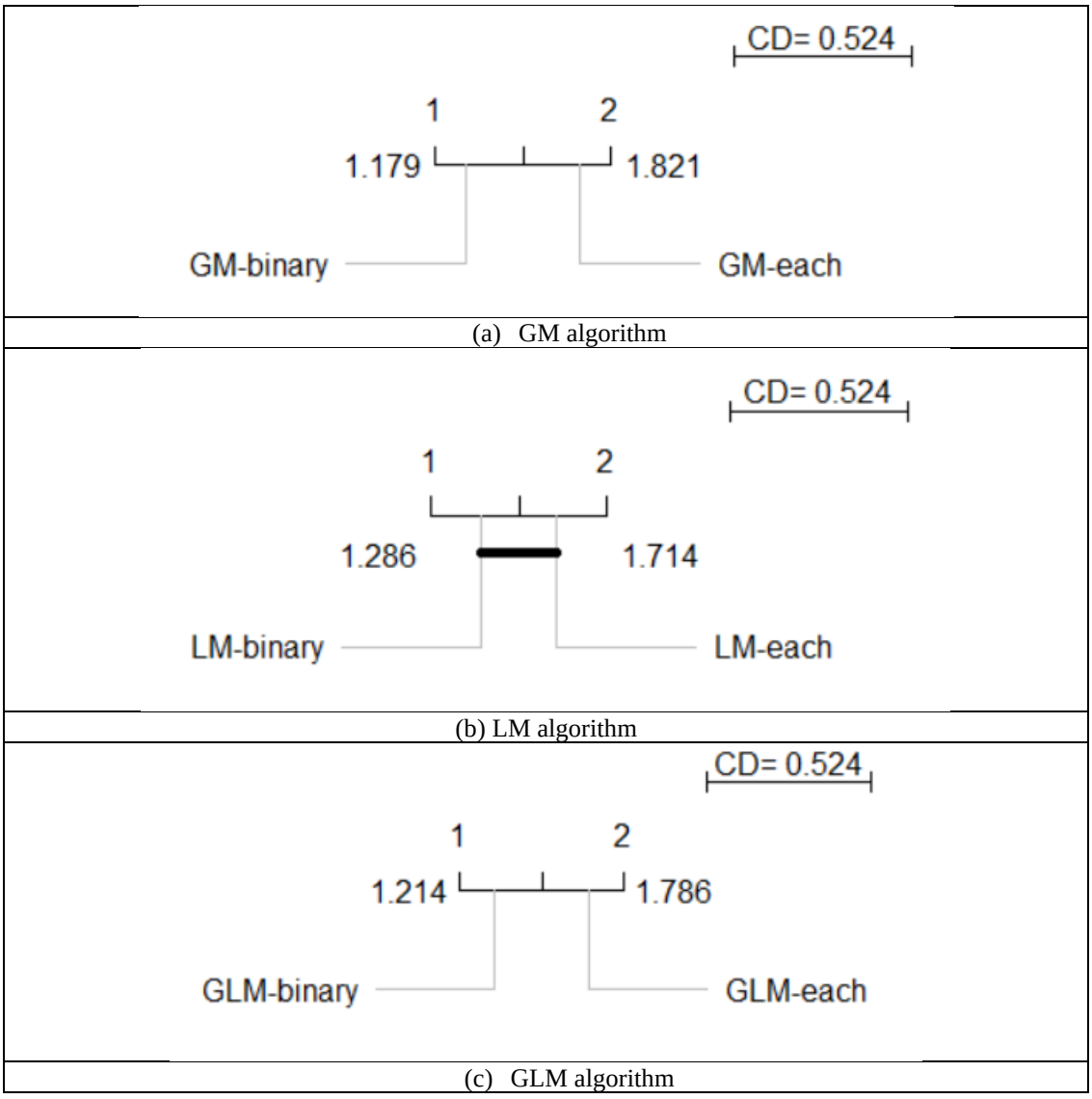


Fig. 5. Critical difference diagram (CDD) between binary vs. each value split: for each algorithm and for the parameter NN

From **Fig. 6**, it is clear that, the best ranked algorithms are achieved by using the binary split when we minimize the GM parameter of the decision tree. But, for no any algorithms, the binary split are stastistically significant (different) than the counterpart of each value split.

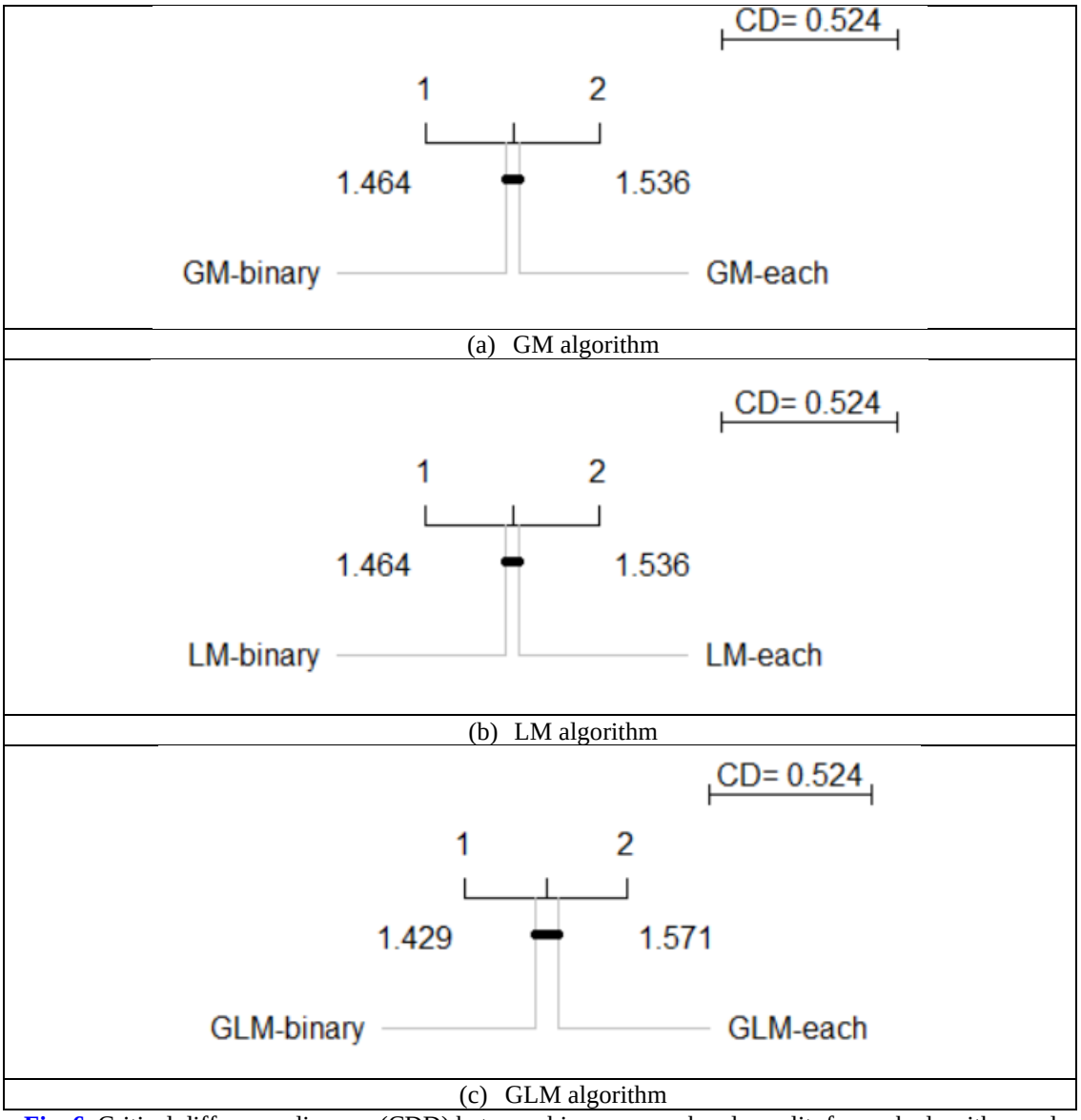
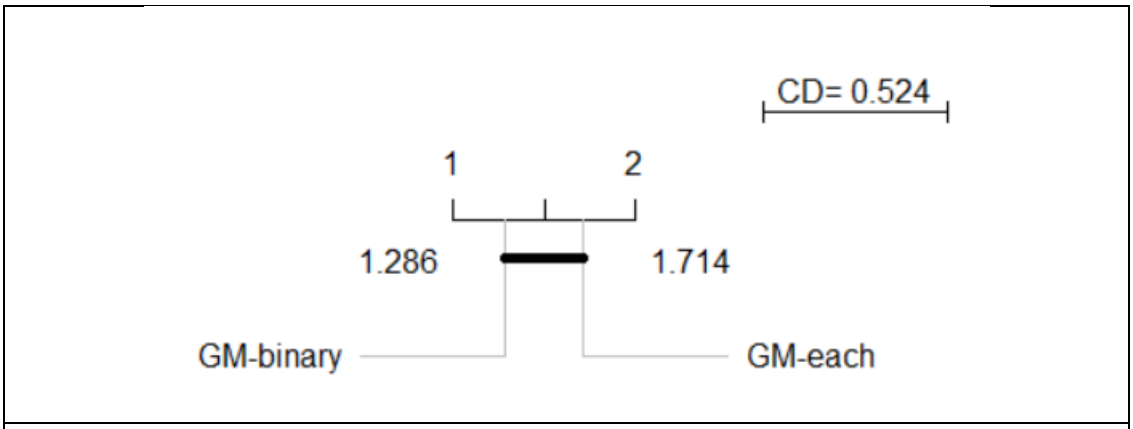
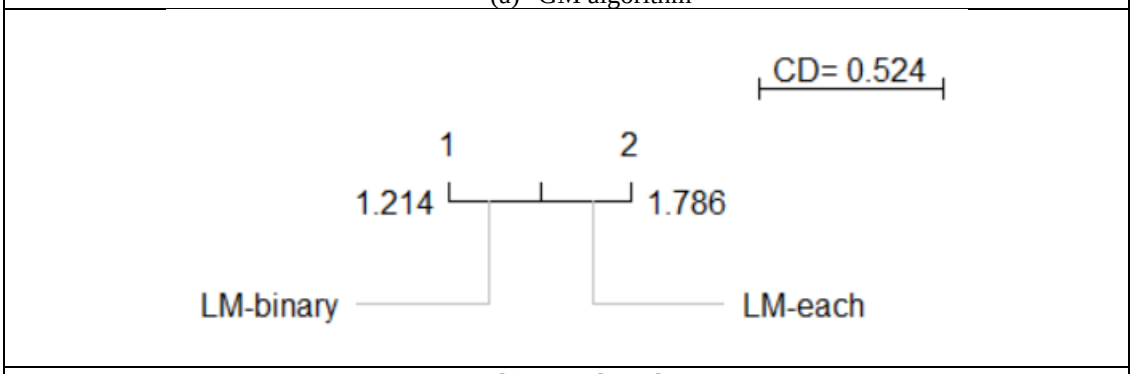


Fig. 6. Critical difference diagram (CDD) between binary vs. each value split: for each algorithm and for the parameter GM

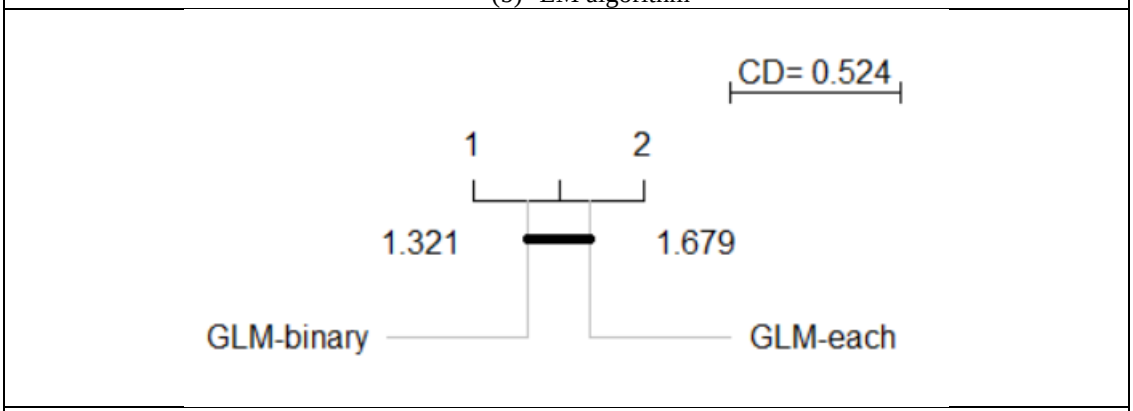
From **Fig. 7**, it is clear that, the best ranked algorithms are achieved by using the binary split when we minimize the LM parameter of the decision tree. Only for LM algorithm (**Fig. 7(b)**), the binary split are stastistically significant (different) than the counterpart of each value split.



(a) GM algorithm



(b) LM algorithm



(c) GLM algorithm

Fig. 7. Critical difference diagram (CDD) between binary vs. each value split: for each algorithm and for the parameter LM

Therefore, these results clearly show the advantages of the binary splits.

5. Conclusion

This paper demonstrates the application of BOO of global and local misclassification rate vs. number of nodes for many decision tables from UCI Machine Learning Repository. The

results are interestingly enough to see that number of nodes, global and local misclassification rates are much smaller for the case of binary splits in comparison with each-value splits. Also it is obvious that GLM algorithm performs well in comparison with other two GM and LM algorithms since it produces smaller NN and reasonable global and local misclassification rates. In future, it is possible to expand the designed algorithms to apply in multi-label decision tables [2, 6]. Furthermore, in the future the study regarding how to limit the number of branches of the constructed DAG will be carried out to overcome the problem of working with the big data.

References

[1] AbouEisha, H., Amin, T., Chikalov, I., Hussain, S., Moshkov, M., *Extensions of Dynamic Programming for Combinatorial Optimization and Data Mining*, Intelligent Systems Reference Library, Springer, vol. 146, 2019. Article (CrossRef Link)

[2] Alsolami, F., Azad, M., Chikalov, I., Moshkov, M., *Decision and Inhibitory Trees and Rules for Decision Tables with Many-valued Decisions*, Intelligent Systems Reference Library, vol. 156. Springer, 2020. Article (CrossRef Link)

[3] Azad, M., Chikalov, I., Moshkov, M., "Decision trees for knowledge representation," in *Proc. of Ropiak, K., Polkowski, L., Artiemjew, P. (Eds.), 28th International Workshop on Concurrency, Specification and Programming, CS&P 2019, Olsztyn, Poland*, September 24–26, 2019. Article (CrossRef Link)

[4] Dua, D., Graff, C., *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, 2019. http://archive.ics.uci.edu/ml

[5] A. Alkhalid, T. Amin, I. Chikalov, S. Hussain, M. Moshkov, and B. Zielosko, "Dagger: A tool for analysis and optimization of decision trees and rules," in *Proc. of Computational Informatics, Social Factors and New Information Technologies: Hypermedia Perspectives and Avant-Garde Experiences in the Era of Communicability Expansion. Blue Herons Editions, Bergamo, Italy*, pp. 29-39, 2011. Article (CrossRef Link)

[6] Azad, M, *Decision and Inhibitory Trees for Decision Tables with Many-Valued Decisions*, Doctoral Dissertation, King Abdullah University of Science & Technology, Thuwal, Saudi Arabia), 2018. Retrieved from http://hdl.handle.net/10754/628023

[7] M. Azad, I. Chikalov, S. Hussain, and M. Moshkov., "Multi-pruning of decision trees for knowledge representation and classification," in *Proc. of 3rd IAPR Asian Conference on Pattern Recognition, ACPR 2015, Kuala Lumpur, Malaysia, November 3-6, 2015*, pp. 604-608, 2015. Article (CrossRef Link)

[8] Moret, B.M.E., Thomason, M.G., Gonzalez, R.C., "The activity of a variable and its relation to decision trees," *ACM Trans. Program. Lang. Syst.*, 2(4), 580–595, 1980. Article (CrossRef Link)

[9] Hyafil, L.,Rivest, R.L., "Constructing optimal binary decision trees is NP-complete," *Inf. Process. Lett.*, 5(1), 15–17, 1976. Article (CrossRef Link)

[10] Chikalov, I., Hussain, S., Moshkov, M., "Totally optimal decision trees for Boolean functions," *Discret. Appl. Math.*, 215, 1–13, 2016. Article (CrossRef Link)

[11] Riddle, P., Segal, R., Etzioni, O., "Representation design and brute-force induction in a Boeing manufacturing domain," *Appl. Artif. Intel.*, 8, 125–147, 1994. Article (CrossRef Link)

[12] Boryczka, U., Kozak, J., "New algorithms for generation decision trees – ant-miner and its modifications," *Abraham A., Hassanien A.E, de Leon Ferreira de Carvalho A.C.P., Snásel V. (eds.) Foundations of Computational Intelligence – Volume 6: Data Mining*, pp. 229–262, 2009. Article (CrossRef Link)

[13] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., *Classification and Regression Trees*, Wadsworth and Brooks, Monterey, CA, 1984. Article (CrossRef Link)

[14] Breitbart, Y., Reiter, A., "A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic boolean functions," *Acta Inf.*, 4, 311–319, 1975. Article (CrossRef Link)

[15] Chai, B., Zhuang, X., Zhao, Y., Sklansky, J., "Binary linear decision tree with genetic algorithm," in *Proc. of 13th International Conference on Pattern Recognition, ICPR 1996, Vienna, Austria, August 25–29*, vol. 4, pp. 530–534, 1996. Article (CrossRef Link)
[16] Garey, M.R., "Optimal binary identification procedures," *SIAM J. Appl. Math*, 23(2), 173–186, 1972. Article (CrossRef Link)
[17] Heath, D.G., Kasif, S., Salzberg, S., "Induction of oblique decision trees," in *Proc. of Bajcsy R. (ed.) 13th International Joint Conference on Artificial Intelligence, IJCAI 1993, Chambéry, France, August 28–September 3, 1993, Morgan Kaufmann*, pp. 1002–1007, 1993. Article (CrossRef Link)
[18] Martelli, A., Montanari, U., "Optimizing decision trees through heuristically guided search," *Commun. ACM*, 21(12), 1025–1039, 1978. Article (CrossRef Link)
[19] Riddle, P., Segal, R., Etzioni, O., "Representation design and brute-force induction in a Boeing manufacturing domain," *Appl. Artif. Intel*, 8(1), 125–147, 1994. Article (CrossRef Link)
[20] Schumacher, H., Sevcik, K.C., "The synthetic approach to decision table conversion," *Commun. ACM*, 19(6), 343–351, 1976. Article (CrossRef Link)
[21] Pawlak, Z., *Rough Sets — Theoretical Aspects of Reasoning About Data*, Kluwer Academic Publishers, Dordrecht, 1991. Article (CrossRef Link)
[22] Pawlak, Z., Skowron, A., "Rudiments of rough sets," *Inf. Sci.*, 177(1), 3-27, 2007. Article (CrossRef Link)
[23] Moshkov, M., "Time complexity of decision trees," *Peters, J.F., Skowron, A. (eds.) Trans. Rough Sets III. Lecture Notes in Computer Science*, vol. 3400, pp. 244–459, 2005. Article (CrossRef Link)
[24] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, Dec. 2006. Article (CrossRef Link)

Mohammad Azad / https://orcid.org/0000-0001-9851-1420
Mohammad Azad obtained Ph.D. in Computer Science from King Abdullah University of Science and Technology. Currently, he is working as an assistant professor in Jouf University, Sakaka, Saudi Arabia at the college of Computer and Information Sciences. Dr. Azad is author and/or coauthor of one research book published by Springer and over 35 journal and conference papers. He is also working as a reviewer of many international journals.